LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

SÓLO PROGRAMADORES

Revista especializada para usuarios de PC

AÑO 3. Nº 22
250 PTAS.
ARGENTINA 9'50 $
CHILE 3000 $
PORTUGAL 1500$

PROGRAMACIÓN DE MODOS X

REVISIÓN DEL NÚCLEO DE LINUX

ESTUDIO DEL COPROCESADOR 387

Y además:

- CÓMO PROGRAMAR UNA DEMO
- REDES LOCALES
- GRANDES SISTEMAS

CURSOS PRÁCTICOS DE:

- Creación de un compilador
- Formatos de sonido
- Visual Basic 4.0

Netscape-[Java: Programing for the Internet]

File Edit View Go Bookmarks Options Directory Window Help

REDES • INTERNET • COMUNICACION • PAGINAS WEB

ESCRITA por desarrolladores para desarrolladores

CD-ROM DE REGALO
COMPILADORES Y TUTORIALES DE ADA

TOWER COMMUNICATIONS, S.R.L

JAVA: REVOLUCIÓN EN INTERNET



La tarjeta de respuesta ya ha sido utilizada.

Network Management
Unix
Parallel Systems
Cliente/Servicio
Internet
Objetos
Bases de datos relacionales
N
W
E
S

NO se preocupe. Envíenos por correo electrónico su nombre, dirección y función a:
SDOEMEA@UK.IBM.COM
o llámenos al 900 100 400

Cómo lograr y mantener una posición líder en el vertiginoso negocio del desarrollo de software.

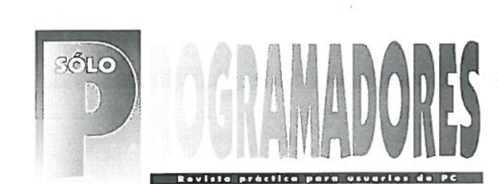
Para desarrollar programas se necesitan buenas ideas así como los conocimientos y las herramientas necesarios para tranformarlas en soluciones excepcionales. La clave para llegar y mantenerse en una posición destacada es el acceso fácil a la información necesaria. Por eso IBM ha creado D_Mail: un nuevo servicio de información para empresas y profesionales desarrolladores de software.

D_Mail le facilita la más reciente información sobre el impacto potencial de las últimas innovaciones, análisis detallados sobre tecnologías clave, el panorama de oportunidades profesionales para el mundo de las soluciones de software y mucho más. ¡Sin coste para usted!

Sólo tiene que devolvernos cumplimentada la tarjeta adjunta, y nosotros le ayudaremos a trazar el rumbo correcto hacia el futuro.



Soluciones para nuestro pequeño mundo



JUNIO 1996. Número 22
SÓLO PROGRAMADORES es una publicación de Tower Communications

Editor
Antonio M. Ferrer Abelló
Director Editorial
Mario de Luis García
Subdirector Editorial
Carlos Doral Pérez
Director Financiero
Francisco García Díaz de Liaño
Director de Producción
Carlos Peropadre
Director de Distribución
Enrique Cabezas García
Directora Comercial
Carmina Ferrer

Director
Mario de Luis García
Redactor Jefe
Carlos Doral Pérez
Coordinador Técnico
Eduardo Toribio Pazos
Editor Técnico
Luis Martin Caballero
Colaboradores
Fernando de la Villa, Fernando J. Echevarrieta, Pedro Antón. Juan M. Martín, Luis Martín, José M. Peco, José C. Remiro, Agustín Guillén, Daniel Navarro, Jorge del Río, Enrique De Alarcón, David Aparicio, M. Jesús Recio, César Sánchez, Santiago Romero, Miguel Cubas, Vicente Cubas

Jefe de Diseño
Fernando García Santamaría
Maquetación
Clara Francés
Tratamiento de Imagen
María Arce Giménez
Servicios Informáticos
Digital Dreams Multimedia, S.L.
Secretaría de Redacción
Rosa Arroyo
Suscripciones
Erika de la Riva
Redacción, Publicidad y Administración
C/ Aragoneses, 7
28100 Pol. Ind. Alcobendas (MADRID)
Telf.: (91) 661 42 11 / Fax: (91) 661 43 86
Filmación
Filma Dos S.L.
Impresión
- G. Reunidas. - Madrid
Distribución
SGEL

Depósito legal: M-26827-1994
ISSN: 1134-4792



LOS FINALES DE JUNIO

Gracias a las cartas llegadas a la redacción, sabemos que muchos de nuestros lectores son estudiantes, la mayoría de carreras técnicas. La verdad es que, esta circunstancia conlleva que el presente mes de Junio sea uno de esos meses fatídicos, en los que se consumen ingentes cantidades de café, se duermen muy pocas horas, y hacer fotocopias se convierte en una misión casi imposible.

La verdad es que, este es el mes en el que la comunidad de estudiantes invade todos esos lugares que hasta ahora eran meras representaciones de los mayores desiertos del mundo. Lugares como las bibliotecas se transforman en auténticos campos de batalla en los que la posesión de algunos libros específicos pueden considerarse un privilegio superior al de una primitiva de 6. Otra de las actividades cuya dificultad raya los límites de lo humano en estas fechas es la realización de fotocopias. Para ser sinceros, este tipo de empresas hace su verdadero Agosto en estos días, gracias al elevado tráfico de apuntes que se produce.

Para que engañarnos. Casi todos hemos pasado por esta situación, por lo que sabemos lo que se hecha ahora de menos el tiempo empleado en aprender cómo acceder a determinado puerto desde nuestro programa en C, en vez de dedicarnos a comprender el funcionamiento de un acelerador de partículas. Generalmente, hemos invertido nuestro tiempo a aprender todo aquello relacionado con la informática y la programación, que en realidad no tenía absolutamente nada que ver con las asignaturas de cualquiera de las carreras del ámbito universitario español. De todas formas, continuando con esta línea de sinceridad, debo decir que la mayoría de nosotros no nos arrepentimos de esta "supuesta pérdida de tiempo". En realidad, hoy por hoy, la mejor opción es la combinación más o menos equitativa del tiempo en ambas actividades. Un título universitario siempre puede ayudar a conseguir un puesto de trabajo, pero los conocimientos añadidos sobre informática o programación pueden suponer el empujón definitivo para que al final seamos contratados.

De cualquier forma, lo mejor de todo esto es el tiempo dedicado a aquello que es nuestra pasión, la programación. Gracias a ella, todos los meses coincidimos en estas páginas, con las ilusiones renovadas, con el fin, por nuestra parte, de ver cómo llenamos las páginas con lo mejor de la gran cantidad de artículos que recibimos, y por la vuestra, aprender y seguir los cursos de los diversos lenguajes y aspectos de la programación.

Desde la redacción, casi damos por hecho que las páginas de esta revista serán leídas en los pequeños momentos de asueto con los que contará la comunidad de estudiantes que tan fielmente nos lee. Incluso, en algunos casos, la lectura del número se realizará casi junto con las del mes que viene, momento especialmente indicado para recuperar esas horas dedicadas al estudio que no se han podido dedicar a lo que verdaderamente nos gusta. La programación. En fin, sea cuando sea, esperamos que este número os resulte tan atractivo como todos los anteriores, o, a ser posible, un poco más. Mucha suerte a todos aquellos que os enfrentáis a las duras horas de estudio y exámenes. Al resto, simplemente animo, que las vacaciones de verano ya están a la vuelta de la esquina. Nos leemos.

SUMARIO

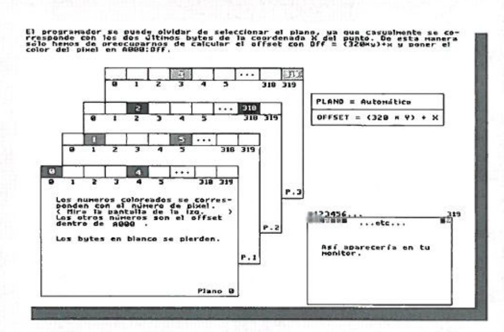
NÚMERO 22

EN PORTADA



La revolución en Internet es el Java que viene, y Sólo Programadores quiere permanecer en la vanguardia de las nuevas tecnologías, comenzando dentro de la sección Web un nuevo curso dedicado a este más que nuevo lenguaje nueva tecnología

DESTACADO



Sabemos que muchos de nuestros lectores demandan más secciones de bajo nivel, Junio es un buen mes para empezar a estudiar a fondo los famosos modos no documentados de la VGA, conocidos popularmente como Modos X. También destacamos en este número un artículo dedicado al coprocesador 387. Por supuesto todos nuestros cursos siguen avanzando, y la mayoría de los conocimientos ofrecidos en los diversos cursos tienen una gran demanda en el mercado laboral

Herramienta de auditoría para empresas

Microsoft Ibérica ha anunciado Microsoft Auditing Resource Kit, la primera herramienta de auditoría de software para empresas. Esta herramienta permite el diseño y la elaboración de auditorías de software para asegurar que el software instalado en los ordenadores es utilizado correctamente. Microsoft Auditing Resource Kit entra dentro del plan de acciones de Microsoft para evitar el uso de software ilegal dentro de las empresas y está dirigida a desarrolladores de software independientes, consultorías, auditorías, técnicos de sistemas y otros grupos de la industria de la programación.

La herramienta incluye software y código fuente necesario para realizar auditorías pequeñas y medianas, hasta un máximo de 500 ordenadores. En el caso de auditorías mayores o personalizadas, el kit proporciona las especificaciones para estos informes y el código fuente necesario con objeto de ampliar el alcance de la auditoría.

Microsoft Auditing Resource Kit permite la obtención de información completa (versión, fabricante y nombre del producto) de todas las aplicaciones para Windows.

Para más información:
S.E.I.S. (Spanish Executive Information Service)
José González Rivera
Tel: (91) 308 43 15
Fax: (91) 310 38 76

Nuevos adaptadores Ethernet PCI

Olicom ha reforzado su posición en el mercado con la presentación del adaptador Ethernet PCI/II 10. Este adaptador proporciona una solución PCI rápida y fiable para los clientes y servidores que ejecutan aplicaciones con amplia utilización de la CPU, como pueden ser las que integran voz y vídeo. El adaptador incorpora la característica de Auto Negociación que, en entornos Ethernet conmutados, detecta y ejecuta automáticamente la operación full duplex (envío y recepción simultánea).

El Ethernet PCI/II 10 es uno de los pocos adaptadores Ethernet disponibles en la actualidad que soporta Adaptative Performance Optimization, una nueva tecnología que mejora considerablemente el rendimiento al proporcionar prestaciones automáticas que se ajustan a la carga del bus PCI del PC. El adaptador es Plug-and-Play y es totalmente compatible con todos los estándares más importantes de la industria, que incluyen la última versión del bus local PCI (revisión 2.1), Ethernet IEEE 802.2/802.3. También soporta el estándar DMI (Desktop Management Interface) y el agente de sobremesa SNMP.

Están disponibles drivers para los principales sistemas operativos en red. Además, soporta los sistemas clientes basados en DOS, OS/2, Windows para Grupos, Windows NT y Windows 95.

Para más información:
Olicom Ibérica
Jacobo de Cal
Tel: (91) 372 98 14
Fax: (91) 372 96 45

Soluciones de plataforma para el Ultra Low-Power Intel 486

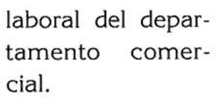
Intel anunció recientemente la disponibilidad a nivel mundial de una solución completa de plataforma que soporta el procesador Ultra Low-Power Intel486. La solución de plataforma de Intel incluye, además de los microprocesadores Intel486 SX/GX, una tarjeta de evaluación para el desarrollo de sistemas de baja potencia, conjuntos de chips, BIOS de sistema y herramientas. El procesador Ultra Low-Power Intel486 SX incluye mejoras a nivel de arquitecturas que permiten el control automático del reloj integrado en el chip. Estas mejoras pueden lograr hasta un 50 por ciento de reducción de consumo de este procesador cuando se compara con los procesadores Intel 486 SX basados en la tecnología SL. El procesador Ultra Low-Power Intel486 está disponible en dos configuraciones de bus externo: una versión de 32 bits que presenta frecuencias operativas de 25 o 33 MHz y una versión de bus externo de datos de 16 bits con frecuencias operativas entre los 16 y los 33 MHz. Los procesadores vienen en encapsulado TQFP de 176 pines, o bien en forma de matriz como un producto Intel SmartDie.

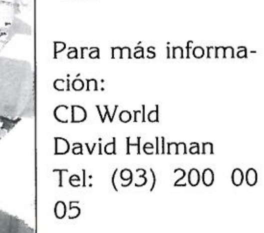
Para más información:
Intel Corporation Iberia
Tel: (91) 308 25 52
Fax: (91) 310 54 60
World Wide Web: http://www.intel.com

Atención Digital al Cliente

34 Telecom, empresa del grupo CD World, ha desarrollado un sistema de procesamiento de llamadas con la habilidad de convertir la voz humana en información de formato digital y ser almacenada. El sistema recibe el nombre de ADAC (Atención Digital Al Cliente) y posibilita que un ordenador emule de forma ágil y dinámica las actividades realizadas en una llamada telefónica. De este modo se pone al alcance de las pequeñas y medianas empresas la tecnología de respuesta por voz para conseguir la funcionalidad exigida por las grandes compañías. ADAC multiplica la capacidad de procesar llamadas de forma sencilla con un coste muy bajo.

El sistema ADAC incluye las siguientes características: correo electrónico de voz, audiotex, fax bajo encargo, telemarketing y respuesta interactiva de voz (HIVR). Además, ADAC permite el paso de las líneas a través de una centralita, el acceso directo a una BBS, servicios de atención al cliente automatizados, y líneas de recepción y registro de pedidos, permitiendo optimizar la dedicación laboral del departamento comercial.





Para más información:
CD World
David Hellman
Tel: (93) 200 00 05



XX Jornadas Técnicas de Relaciones Públicas

Durante los días 22, 23 y 24 de mayo tuvo lugar en el auditorio Mapfre Vida de Madrid la vigésima edición de las Jornadas Técnicas de Relaciones Públicas del Centro Español de Nuevas Profesiones, bajo el título "Nuevas Tecnologías: Tan lejos como tú quieras". Estas jornadas técnicas se vienen realizando ininterrumpidamente desde hace 19 años, sin ánimo de lucro, por los alumnos del último curso de la Escuela de Relaciones Públicas del centro y gracias a la colaboración desinteresada de diversas empresas y organismos.

Este año se centraron en un tema de máxima actualidad, como son las nuevas tecnologías. Las XX Jornadas contaron con la ayuda de destacados profesionales del sector como D. Javier Bardaji (Director de Marketing de El Mundo), D. Ángel Para (Director de Expansión y Desarrollo de Hispaservices, S.A.) y D. José L. Tejerina (Director de tecnología de Retevisión), entre otros. Algunos de los asuntos tratados fueron la multimedia, el CD-ROM, la televisión por cable, la telefonía móvil e Internet.

Para más información:
Secretaría de las XX Jornadas Técnicas
Centro Español de Nuevas Profesiones
Pza. Del Conde Valle Suchil, 2
28015, Madrid

Olicom anuncia el Conmutador Token-Ring CrossFire

Olicom ha anunciado el conmutador Token-Ring CrossFire, capaz de incrementar de forma económica y fácil de implementar, las prestaciones de las redes Token-Ring. A la vez, el conmutador Token-Ring CrossFire permite a los administradores de red implementar de manera flexible ATM a 155 Mbps para el transporte backbone de alta velocidad.

CrossFire proporciona de 8 a 12 puertos, cada uno de los cuales soporta un segmento Token-Ring de 1 o 16 Mbps dedicado o compartido vía cableado UTP/STP y conectores RJ-45. Se puede utilizar un slot de expansión para añadir un Módulo de Expansión Universal para cuatro puertos Token-Ring, dos puertos fibra Token-Ring o un puerto ATM 155 Mbps que soporta emulación LAN. Cada puerto Token-Ring soporta hasta 1.700 conexiones activas para un máximo de 6.000 totales por conmutador.

El conmutador Token-Ring CrossFire está disponible en volumen al precio de 5.600 dólares.

Para más información:
Olicom Ibérica
Jacobo de Cal
Tel: (91) 372 98 14

S-Designor 5.0 PowerSoft

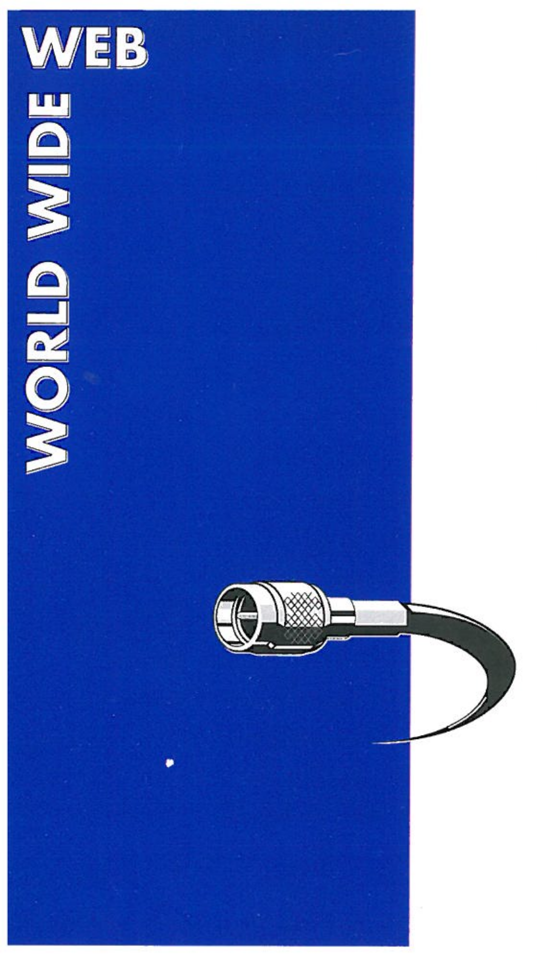
PowerSoft ha presentado la nueva gama de herramientas S-Designor 5.0 de análisis y diseño sobre plataformas cliente/servidor, que facilita la creación de bases de datos y aplicaciones de alta calidad.

La versión 5.0 de S-Designor incluye cuatro módulos separados y compatibles que permiten crear, tanto a particulares como a equipos, un juego de herramientas a la medida de sus necesidades con un coste ajustado. Estos módulos son: S-Designor 5.0 DataArchitect para diseño y construcción de bases de datos, S-Designor 5.0 ProcessAnalyst para análisis de procesos, S-Designor AppModeler para generación de aplicaciones, y S-Designor 5.0 MetaWorks para capacidades avanzadas de diccionario multiuso.

S-Designor posee una estructura modular para ajustarse con facilidad a las necesidades concretas de cada grupo de trabajo. Los módulos DataArchitect y AppModeler se pueden adquirir como productos individuales. Como módulos opcionales, ProcessAnalyst sirve de complemento a DataArchitect, mientras que MetaWorks ofrece a los otros tres módulos una mayor cantidad de prestaciones de cara al trabajo en equipo. Las nuevas características incluyen un interfaz bajo Windows 95 y una nueva forma de realizar informes de diseño personalizado.

Para más información:
Sybase Iberia
Marcos Carmena
Tel: (91) 302 09 00


Nunca funciona
igual de bien sin
la parte central.



JAVA: LA REVOLUCIÓN INTERNET

Fernando J. Echevarrieta

Todo el mundo ha oído hablar de Java, aunque son menos los que lo han visto en acción. Pocos son ya los que conocen la repercusión que esta nueva tecnología puede tener, desbordando completamente el mundo WWW.

Java está "disfrazado" de Web, pero se trata de algo que va mucho más allá. Como se verá a lo largo del artículo se trata de una nueva tecnología que desde Internet puede provocar un cambio en los sistemas de trabajo e incluso en la misma concepción del ordenador y la informática personal, llevándolos hasta otra dimensión hasta ahora sólo conocida en la literatura de ficción. En definitiva, podemos estar asistiendo a la mayor revolución tecnológica desde la aparición del microprocesador. Por ello, es necesario separarse un momento de los aspectos puramente técnicos con el fin de que "los árboles no impidan ver el bosque" y diponer de una visión de conjunto antes de adentrarse en las especificaciones del lenguaje.

UN MOMENTO DE CAMBIO

Nos encontramos en un período de cambios. Si bien durante el último año las tecnologías relacionadas con el WWW han evolucionado a un vertiginosa velocidad que dejaba atrás todos los borradores de estándares, en este momento, se está desarrollando un frente común que hace converger los que eran estándares "de facto".

Por una parte el *VRML* (*Virtual Reality Markup Language*, en un principio y *Modeling Language*, posteriormente) aún no tratado en esta serie, aportaba al usuario de WWW la posibilidad de navegar por escenarios 3D que muy aventuradamente se calificaron de realidad virtual. Sin embargo, la especificación 1.0 de VRML únicamente define geometrías, entornos 3D, pero no está dotada de un mecanismo que proporcione semántica a los objetos. Este es el caballo de batalla de los grupos que trabajan en VRML, lo que ha permitido ya varias soluciones distintas para la resolución de colisiones.

Por otra parte entraba en escena *Java*, aportando una interactividad al WWW como no se había imaginado anteriormente e implantando un nuevo lenguaje orientado a objetos independiente de la arquitectura.

Al hilo de todo ello Netscape Communications seguía en una carrera por convertirse en el Microsoft de Internet con el desarrollo de las nuevas versiones 2.x de *Netscape Navigator*, con 4 nuevas mejoras:

1. La ampliación del HTML mediante el uso de *frames*, que empiezan a poblar de ventanas el WWW y el reconocimiento de los *client-side imagemaps* de Mosaic, que también han sido adoptados por el Internet Explorer de Microsoft;
2. El desarrollo de una API de programación que permite el desarrollo de *plug-ins*, aplicaciones integradas en el propio browser, entre las que se encuentra un navegador VRML;
3. La integración de una máquina virtual *Java* en el browser que permite la visualización de páginas que incluyen programas Java;
4. El desarrollo de un nuevo lenguaje de script para su browser denominado



James Gosling, el creador de Java.

JavaScript y que, pese a la similitud del nombre y a la equivocada creencia de mucha gente, poco tiene que ver con Java salvo en la funcionalidad de ejecutar ciertos scripts desde el cliente.

Y ahora, gigantes como Silicon, SUN y SONY, entre otros, se unen para hacer converger estas tecnologías mediante la especificación *Moving Worlds* propuesta para su aprobación como especificación 2.0 de VRML. *Moving Worlds* supone un verdadero sistema de navegación por Internet basado en realidad virtual dotando a VRML de una API de programación para Java y C++ que permite que estos y quizá otros lenguajes en el futuro proporcionen la semántica necesaria a los objetos del mundo 3D.

Por estas razones, será necesario romper la estructura lineal de esta serie, como se anunciaba el mes anterior, para saltar mes a mes de una a otra tecnología.

¿QUÉ ES JAVA?

Java es un lenguaje orientado a objetos desarrollado por SUN Microsystems que ha alcanzando gran popularidad en el WWW. Al menos, éste ha sido su escaparate publicitario ante el gran público, aunque SUN tiene para él otros planes mucho más ambiciosos. La gran ventaja que aporta sobre otros medios de programacion en el WWW como CGI es su interactividad. El sistema define una máquina virtual independiente del hardware, que es capaz de ejecutar programas denominados *applets* que le envía el servidor WWW. De esta forma, es capaz de adaptarse a cualquir posible extensión de los protocolos, a cualquier tipo nuevo de datos ya que junto a estos, recibe el código capaz de procesarlos. Al ejecutarse los *applets* en el interior de la máquina virtual que se sitúa en el cliente, es posible realizar documentos interactivos que se ejecutan como una aplicación más en el escritorio de trabajo, siendo capaces de responder al teclado y al ratón y de gestionar imagen y sonido.

¿CÓMO SE ACCEDE A JAVA?

En el mundo de Java, existen dos puntos de vista: el del usuario y el del programador. Desde el punto de vista del usuario, para acceder a documentos WWW que incluyan applets de Java es necesario disponer de un browser especial con la marca "*Java Compatible*". En el momento de escribir este artículo existen dos: *Netscape Navigator* en sus versiones 2.x y *HotJava* de SUN. Existe también otra forma de disfrutar de programas Java sin necesidad de un browser de Web, mediante un programa llamado *appletviewer*. Ya han aparecido los primeros modelos de ordenadores basados en Java y en un futuro muy próximo se convertirán en algo común.

Desde el punto de vista del programador, para desarrollar programas Java se necesita lo que SUN denomina *JDK* o *Java Development Kit*, que facilita para Solaris, Windows 95 y Windows NT y del cual se ha realizado ya un port para Linux ELF. El JDK incluye un compilador, un depurador y el *appletviewer*.

Existe también un punto de vista intermedio. El del usuario de Internet que realiza sus propias páginas WWW y desea aportarles la funcionalidad y vistosidad que proporciona Java sin tener que aprender este lenguaje. Para ello, cualquier browser "Java compatible" entiende ciertas extensiones nuevas de HTML como la etiqueta *<applet>* que permiten incluir applets desarrollados por teceros en cualquier página HTML. También se han definido nuevas etiquetas para proporcionar parámetros a estos applets.

UN POCO DE HISTORIA Y ANÉCDOTAS

La filosofía de Java es una consecuencia clara de las causas de aparición y evolución del lenguaje. Al menos curioso resulta el hecho de que Java es un lenguaje ideado inicialmente para programar electrodomésticos. En la actualidad, hasta los bienes de electrónica de consumo más simples, como un horno o un aparato de aire acondicionado o como "los chicos de SUN" ponen de ejemplo, una tostadora, se encuentran dotados de microcontroladores o microprocesadores que controlan su comportamiento. Se supone que estos aparatos deben tener una larga vida. Larga, al menos, en comparación con la vida media actual de su software. En una palabra, cada vez más, el software debe ser reemplazable.

Es con este problema con el que se encuentra allá por el año 1990, *James Gosling*, que dirige un pequeño grupo de gente en SUN Microsystems. Su dilema surge de que no encuentran ningún lenguaje apropiado al efecto. Lenguajes como C o C++ deben ser compilados para un chip y, si se cambia el chip, todo el software debe compilarse de nuevo. Este problema es especialmente acusado en el campo de la electrónica de consumo. La aparición de un chip más barato y, generalmente, más eficiente, conduce inmediatamente a los fabricantes a incluirlo en las nuevas series de sus cadenas de producción por pequeña que sea la diferencia de precio ya que, multiplicada por la tirada masiva de los aparatos, supone un ahorro considerable. Así es como este año 1990, James Gosling decide crear un nuevo lenguaje de programación más apropiado que, en el futuro, se llamaría Java.

El primer proyecto en el que se aplicó el lenguaje recibió el nombre de *proyecto Green* y consistía en un sistema de control completo de los aparatos electrónicos y el entorno de un hogar. Para ello, se contruyó un ordenador experimental denominado *7 (*star seven*). El sistema presentaba una interfaz basada en la representación de la casa de forma animada y el control se llevaba a cabo mediante una pantalla sensible al tacto. En el sistema aprecia un dibujo animado denominado *Duke* que se convertiría en la actual mascota de Java. Posteriormente, el lenguaje se aplicó a un otro proyecto denominado *VOD: Video On Demand* en el que se empleaba como interfaz para la televisión interactiva.



Duke, la mascota animada de Java.

Ninguno de estos proyectos se convirtió nunca en un sistema comercial, pero fueron desarrollados enteramente en Java, por lo que sirvieron como bautizo de fuego del lenguaje, para detectar sus carencias y para mejorarlo. El lenguaje aún sigue en evolución en fase de beta, aunque su filosofía principal ha quedado asentada.

JAVA LLEGA AL WWW

Hasta aquí todo parece muy interesante, pero no parece tener relación con el WWW. Y es que los dos mundos no se encuentran hasta que, en 1993, *Arthur Van Hoff* y *Shaimi Shaiao* se unen al proyecto. El lenguaje HTML había nacido ya dos años antes, en 1991, de manos de *Tim Berners-Lee* (con lo que, curiosamente, resulta ser posterior a Java) y este año 1993 ya se estaba desarrollando de forma gráfica (repasar [Echeva-1]). Por aquel entonces nace la especificación 2.0 de HTML y el interfaz CGI ([Echeva-2]) y con ellos los forms ([Echeva-3], [Echeva-4]), que dotan de una realimentación limitada al sistema. Y es en este entorno en el que a los nuevos miembros del proyecto les surge la idea de introducir Java en la tela de araña. Con este fin, desarrollan un browser de WWW denominado *WebRunner* que, por motivos comerciales cambiaría su nombre posteriormente al de *HotJava*. El anuncio oficial se realizaría en la *SunWorld Conference* en San Francisco en Mayo de 1995, aunque el autor del presente artículo ya tuvo el honor de acceder por primera vez al sistema en un exhibit durante la *Third International WWW Conference* en Darmstad, Alemania, un mes antes. Allí provocó tal expectación que el mismo James Gosling hubo de realizar una "oficiosa" presentación fuera de programa el último día, evento que ya fue cubierto como primicia en Sólo Programadores ([Echeva-5]). Pero fue tras la presentación oficial en San Francisco cuando Marc Anderseen, vicepresidente y co-fundador de Netscape Communications anunció que la versión 2.0 de Netscape Navigator soportaría applets de Java, promesa que, en efecto, se ha llevado a cabo.

Como es costumbre en Internet, han surgido numerosas bromas desde entonces acerca del nombre de *JAVA* afirmando que significa *Just Another Vague Acronym*, del mismo modo que *PCMCIA* significa *People Canít Memorize Computer Idiot Acronyms*. El equipo de desarrollo de Java afirma que esto no es así, que inicialmente se denominó *OAK* porque Gosling veía un roble desde su ventana y que cuando descubrieron que ya había un lenguaje con ese nombre se quebraron la cabeza hasta que, un día, durante una visita a una cafetería, a alguien se le ocurrió el nombre de Java, razón por la que el logo de Java y de la marca *Java Compatible* es una humeante taza de café.

CARACTERÍSTICAS GENERALES DEL LENGUAJE

El lenguaje Java es un lenguaje de alto nivel que presenta un gran parecido con C++, al menos, en la sintaxis, aunque semánticamente es prácticamente idéntico a ADA.

Uno de los objetivos fundamentales era el desarrollo de un lenguaje potente pero, a la vez, lo más simple posible. Asimismo se buscaba evitar que el lenguaje pudiera ser una fuente de error en la programación. Por ello, se han eliminado todas las características de los lenguajes de alto nivel que en SUN consideraban "accesorias". Con ello, si bien se ha perdido algo de libertad al programar, por ejemplo, en el manejo de la memoria, se ha conseguido un lenguaje más rápido de aprender que los mencionados, incluso si no se conoce la metodología de orientación a objetos. Entre las caracteristicas que se han considerado innecesarias se encuentran la aritmética de punteros, los ficheros de cabeceras, la sobrecarga de operadores, el uso de un preprocesador, las estructuras unión y los arrays multidimensionales.

Se trata de un lenguaje orientado a objetos. En Java todo son objetos con una única excepción: los tipos simples (ej. números); aunque también proporciona objetos para todos los tipos simples de tal forma que se puedan implementar como clases. El código de un programa Java se encuentra organizado en clases. Cada clase define un conjunto de métodos que determinarán el comportamiento de los objetos. Java está dotado de herencia de propiedades, por lo que las clases se estructuran en una jerarquía en cuya raíz se encuentra la clase *Object*. Esta herencia es simple, es decir, cada clase únicamente puede heredar propiedades de otra única clase. El equipo de desarrollo de Java justifica la no necesidad de utilización de herencia múltiple y su mayor complejidad mediante una serie de argumentos que se verán en su momento.

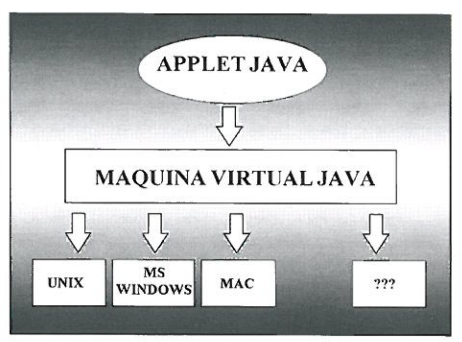


Con este símbolo se marcan las páginas WWW que contienen applets de Java

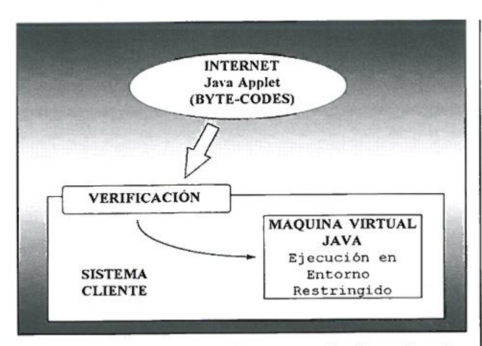
El lenguaje dispone en principio de tipado estático, por lo que es necesario definir el tipo de todos los objetos. Sin embargo, todos los objetos disponen también de un tipado dinámico, lo que permite que un programa actúe de forma distinta con objetos de tipos distintos.

Otra de las características del lenguaje es que soporta la concurrencia a través de *threads*. Así, un *thread* puede encargarse de los cálculos que lleven más tiempo a la vez que otro se encarga de atender al usuario. Si el S.O. que lo sustenta soporta sus propios *threads*, se trata de mapear los de Java sobre los de éste. Otra ventaja de ello es las prestaciones mejoran en un sistema multiprocesador. Hoy por hoy, Java es 20 veces más lento que C aunque aún así es bastante más eficiente que la mayoría de los lenguajes de shell. El equipo de desarrollo de Java anuncia que con futuros generadores de código podrá disponer de una eficiencia comparable a C.

Es el lenguaje y no el programador el que lleva a cabo el control de la memoria al contrario que ocurre en otros lenguajes como C. Esto se ha realizado así



Java alcanza una independencia de la arquitetura definidiendo una máquina virtual



El problema de los virus se elimina desde Java mediante una verificación de los byte-codes y la ejecución en un entorno restringido.

por diversos motivos. En primer lugar se trata de evitar errores de programación. En C y C++ el programador debe ser responsable de liberar la memoria que no va a volverse a utilizar. Java esta dotado de un *garbage collector* (recogedor de basura), encargado de realizar esta función. En segundo lugar, se pretende que el lenguaje sea robusto y un programa no pueda bloquear el sistema. Así la máquina virtual supervisa cada acceso a memoria y, en caso de que se detecte un error, produce una excepción. En tercer lugar, se ha cuidado especialmente la seguridad, lo que ha inducido a eliminar el uso de punteros.

PRINCIPALES INNOVACIONES

Mediante una perspectiva un tanto amplia se podría decir que Java es un lenguaje a la vez compilado e interpretado. Se podría considerar interpretado ya que exige la presencia de la máquina virtual Java que ejecute, interprete, el programa; aunque en realidad, se trata de un lenguaje compilado. El código fuente Java se compila para obtener un código ejecutable denominado *fichero de clases* en forma de *byte-codes* (¿debería traducirse como códigos-byte?). Los *byte-codes* son estructuras de información análogas a las instrucciones-máquina con la salvedad de que, según afirma SUN, no son específicas de ninguna arquitectura.

La principal baza con que cuenta SUN en Java es la independencia del lenguaje respecto a la arquitectura. Por una parte cuenta con sistema propio de representación de la información: un entero son 32 bits en un PC y 32 bits en una SPARC y, además, ordenados de la misma forma. Por otra parte, es portable a nivel de ejecutables (*byte-codes*) sin necesidad de recompilación cuando se cambia de un sistema a otro.

Para acceder al S.O. de soporte, la máquina virtual Java deber proporcionar la siguiente librería de clases:

java.lang: Clases generales del lenguaje, como Object, String, Exception, Error...
java.util: Clases de utilidades, como Properties, Vector...
java.io: Entrada y salida basda en streams. Acceso al sistema de ficheros
java.net: Clases para comunicaciones. Acceso a bajo nivel (sockets) y a alto nivel (URLs)
java.awt: Abstract Window Toolkit.
java.applet: Soporte de applets (programas incrustables).

Por el momento, el lenguaje Java admite librerías de otros lenguajes aunque personal de SUN en España comentó al autor que esta posibilidad estaba destinada a desaparecer en el futuro, entre otras, por razones de seguridad. Con este fin, se permite declarar métodos Java como *native*, para enlazarlos directamente en la máquina virtual con una librería software. realizada en otro lenguaje. En principio, esta capacidad se inhabilita con el software procedente de la red, de nuevo, por motivos de seguridad.

JAVA VS CGI

Aunque la idea del WWW "se ha quedado pequeña" para Java, es cierto que su rápida difusión ha sido gracias a este sistema. Por ello, pensando únicamente en el ámbito del Web, cabe plantearse la pregunta: ¿Qué supone Java respecto a CGI?

Java y CGI suponen dos filosofías completamente distintas. Mientras Java es un lenguaje de programación, CGI es una interfaz entre programas de aplicación y servidores de información [Echeva-6], por lo que no implica aprender ningún lenguaje nuevo. Con CGI la complejidad se encontraba en el servidor, ahora Java la traslada al cliente. Con esto, se gana en eficiencia desde el punto de vista del "navegante de Internet" ya que, si bien un programa CGI no precisa de tiempo de carga en el cliente, en el caso de Java, el tiempo de carga inicial se ve compensado porque todo el procesamiento se realiza en el propio cliente, sin necesidad de accesos continuos a red cada vez que se desea algún tipo de interactividad como ocurría con CGI. Ventajas adicionales son que no se necesita, por tanto, la existencia de un servidor para la ejecución de programas a través del WWW y, lo que es más interesante, no es necesario disponer de ningún tipo de privilegio para incluir programas Java en páginas WWW. Así, es posible grabar durante una navegación una página con un applet de Java y reproducirla off-line en otro sistema operativo, con otra arquitectura y sin necesidad de servidor. Como contrapartida, se requiere un cliente especial que defina una máquina virtual, mientras que con la filosofía CGI, hasta el más elemental de los browsers funciona debido a que son "engañados" para recibir documentos HTML sin conocer su procedencia. Por último, el único punto donde CGI aventaja a Java en el mundo WWW es su capacidad de integración con software existente en el servidor, como el típico ejemplo de consulta a una base de datos que se proponía en el artículo de introducción a CGI.

Así pues, se podría decir que aunque parece clara la impresionante difusión que Java a tener, siempre quedará, a corto y medio plazo, un lugar para CGI, aunque sólo sea para el administrador del sistema que considera más fácil cambiar la salida de sus scripts para que se adapten a la interfaz CGI, que no reprogramarlos completamente en Java y gestionar las comunicaciones para que operen sobre el servidor desde el cliente.

¿VIRUS EN JAVA?

Al considerar los aspectos de seguridad en un sistema, y más en uno en que el software procede de diferentes lugares de una red, uno de los aspectos que más preocupa es la presencia de virus. Para defenderse de esta posibilidad los *byte-codes* se encuentran fuertemente tipados, lo que permite su verificación. Con este fin, los ficheros de clases disponen de información adicional de verificación de los mismos.

Los programas en un PC llaman a las funciones a través de una dirección. Esta dirección no es otra cosa que un número y cualquier programa puede utilizar cualquier número por lo que no es posible saber a priori qué va a hacer el programa. En Java, en principio se accede

JAVA VS CGI	
CGI	**JAVA**
Ejecución en servidor	Ejecución en cliente
Sin tiempo de espera de carga	Tiempo de espera de carga
Esperas continuas para lograr interactividad	No requiere esperas posteriores a la carga inicial
Precisa de un servidor: no se puede almacenar	No requiere un servidor: se puede almacenar
No requiere aprendizaje de un nuevo lenguaje	Es un nuevo lenguaje que se necesita aprender
Impone metodología	Impone metodología
Nivel de interactividad bajo	Nivel de interactividad muy alto
Funciona con cualquier cliente	Requiere un cliente especial
Integración con sistemas existentes en servidor: alta	Integración con sistemas existentes en servidor: baja

a los métodos y a la variables por nombre. Durante el proceso de verificación se comprueba cuáles son las funciones que van a ser accedidas por el programa. Sólo después de esta verificación se convierten los nombres a direcciones. Por otra parte, el programa Java se ejecuta en un entorno restringido a la máquina virtual Java que se puede configurar para limitar el acceso a los recursos del sistema en cualquier medida.

EL "AGUJERO" DE NETSCAPE

Los temas de seguridad informática van mucho más allá de los virus y siempre podrán aparecer programas o sujetos maliciosos que atenten contra un sistema. Para ello, suelen existir herramientas o metodologías de protección, alguna de las cuales se encuentra presente en algunas máquinas virtuales Java.

Así, uno de los casos que más revuelo ha levantado es el famoso "agujero del Netscape 2.0" que ha motivado la pronta aparición de la versión 2.01. La máquina virtual definida por el Netscape Navigator imponía como condición de seguridad que si un programa Java realizaba una conexión de red, esta podría realizarse únicamente con el servidor del que se cargó el código. Sin embargo, esta comprobación se realizaba atendiendo al nombre de la máquina en lugar de a su dirección IP. Este hecho puede ser aprovechado por hackers que lancen servidores de DNS (que convierten nombres a direcciones IP, consúltese [Echeva-7] en este mismo número) que faciliten direcciones falsas, para realizar conexiones con cualquier máquina presente en la red.

En principio esto puede parecer inocuo, sin embargo, piense el lector que la conexión realizada pertenecería al usuario que ejecuta el cliente y que quizá podria realizarse a una máquina como la propia del cliente con los privilegios del mismo y una vez allí y con su identidad... Aunque el proceso no es tan sencillo, es posible y, lamentablemente, muchas veces se descubren estos agujeros porque uno se da cuenta de que ya no tiene sistema que proteger. Sea como fuere, este problema concreto se debía a un fallo en la máquina virtual Java que implementaba Netscape y no al lenguaje Java. Fallo que ha sido subsanado en la última versión en la que la comprobación se realiza sobre las direcciones IP.

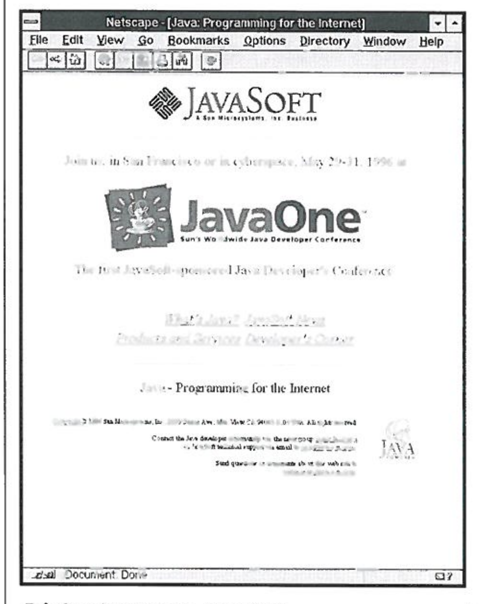
En ocasiones sí puede interesar que el programa Java realice conexiones con terceros, por ejemplo, en entornos de trabajo cooperativo. Por ello, esta opciones suelen ser configurables como es el caso tanto de Netscape como HotJava, que proporciona cuatro niveles de seguridad en el acceso a redes.

FILOSOFÍA DE JAVA

El potencial de Java surge, fundamentalmente de su independencia frente a la arquitectura. En realidad se podría matizar indicando que los byte-codes son instrucciones especificas de la máquina virtual Java, que puede ser implementada sobre cualquier arquitectura. Sea como fuere, lo cierto es que Java proporciona portabilidad de ejecutables mientras que otros lenguajes únicamente ofrecen portabilidad de fuentes y ésta última, además, es teórica, como habrá comprobado cualquier lector que haya tratado de compilar un programa C para una versión de UNIX en otra distinta. Y ni se mencionarán los problemas que surgen al cambiar de S.O.

Sin embargo, el hecho de requerir una máquina virtual hace que tenga algunas de las desventajas de un lenguaje interpretado. No obstante, ¿y si la máquina virtual se convierte en real?. Así, por primera vez surge la idea de un diseño basado en la premisa de que ciertos recursos de internet se encontrarán siempre disponibles. Hasta el momento, los lenguajes se habían centrado en el desarrollo de arquitecturas cliente-servidor basadas en los extremos: en el cliente, en el escritorio de trabajo (*desktop-centric*) y en el servidor (*server-centric*). Mediante la filosofía de Java, la atención se centra en cómo trabaja todo combinado (*network-centric*).

De esta forma, los hábitos de trabajo cambiarían. La idea no es ya que un programa Java sea un applet en una página de Web. Ahora, incluso el propio browser de Web podría ser un applet de Java que se cambiara dinámica y automáticamente en función de las necesidades de comunicación, por ejemplo, al encontrarse con un protocolo no conocido. De este modo, las tornas cambian y Java pasa de ser contenido a continente. Con esta nueva filosofía, el modo de ejecutar aplicaciones también cambiaría. Todo el trabajo se realizaría mediante un acceso On-Line lo que, asimismo, induciría un cambio en la estructura del mercado. Ahora, se podría cobrar por el uso que se hace de las aplicaciones. Se obtendrían de la red las funcionalidades necesarias según se fueran necesitando. Hoy en día, cualquier paquete de software incorpora multitud de funciones complejas de las cuales el usuario medio no llega a utilizar más del diez



Página WWW de JavaSoft

por ciento. Sin embargo, debe pagar por la totalidad. Por otra parte, el precio es constante se haga poco o mucho uso del mismo. Con Java y un sistema basado en red, esto cambiaría. Del mismo modo, no sería necesario pagar y esperar por actualizaciones ya que la red aseguraría que siempre se dispone de la última versión y se cobraría por su uso. Para SUN y otras compañias como Oracle, todo esto son ventajas. El autor opina que el lector debe juzgar por sí mismo si esto es así. Lo cierto es que acertada o no, se abre una nueva vía que puede suponer un giro tecnológico.

JAVA: ¿UNA REVOLUCIÓN?

En este momento la idea de Larry Ellison, presidente de Oracle Corp. de que la informática personal debe pasar por la desaparición del PC que sería relevado por el *NC* (*Network Centric Computer*) comienza a no parecer tan descabellada. Un NC podría ser una máquina real, ya no virtual, que empleara Java como S.O. De este modo, se podrían realizar NCs de bajo precio, unos $500, que no requirieran de grandes cantidades de memoria ni de almacenamiento y ofrecieran la posibilidad de cargar y ejecutar aplicaciones de la red desarrolladas con independencia del microprocesador empleado. Oracle ha apostado por esta idea y en Enero presentó en Chiba, Japón su primer NC basado en un procesador RISC ARM7500 a 50Mhz con 8Mb de RAM, conexión ethernet, telado y un slot de PC. Pero Oracle no es la única: SUN ha anunciado que tendrá su propio NC este mismo año, Apple ha licenciado tecnología para el desarrollo de un sistema Network Centric denominado Pippin e IBM afirma que esta desarollando el suyo.

Aun así surge la duda de por qué debe ser tan caro como $500 un sistema NC. Ben Schneiderman,de la Universidad de Maryland ha diseñado un "*Webtop*" que consiste en una pantalla LCD sensible al tacto mediante la que se puede navegar por el Web y cuyo precio asciende a $100. Aunque es suficiente para cumplimentar formularios Web, es posible ampliarlo mediante un teclado de $30. No es el único en afirmar que el precio actual de los sistemas informáticos, que ronda los $2000, impida que estos lleguen al gran público como lo ha hecho la televisión.

Esta tendencia se encuentra en clara contraposición con la defendida por Microsoft e INTEL que tienden a defender la idea del PC dotándole cada vez de mayor potencia. Para ellos, la red también es un pilar fundamental, pero como herramienta para el PC, en el que enfocan sus objetivos.

Otras personas como Gary Chapman, coordinador del 21st Century Project en la LBJ Scholl of Public Affairs de la Universidad de Austin, Texas, afirman "La mayoría de los usuarios de ordenadores personales no comprenden que la máquina que han adquirido y con la que han luchado para aprender, podría coordinar un hotel mediano o un pequeño hospital". De hecho, el ordenador de a bordo en las naves Apollo era equivalente a un Spectrum, pero Chapman añade: "Los ordenadores personales han llegado a ser tan potentes que estan rozando el límite de lo ridículo y, por mantener los precios altos, la industria no está expandiendo el mercado a nuevos usuarios".

Sin embargo, el problema actual parece ser el ancho de banda: no es probable que la gente confíe en un NC con la Internet actual, dado que no es ni lo suficientemente rápida ni lo suficientemente segura. Por ello, todas las empresas se apresuran en una carrera por conseguir aumentar el ancho de banda. Por el momento, la unión de AT&T Bell Labs, Digital Equipment Corp. y el MIT ha logrado el desarrollo experimental de redes ópticas hasta a 2.4 Gb por segundo, el estándar actual es de 115 Mb por segundo. Quizá ahora Bill Gates, ante la idea de los NCs basados en Java, se arrepienta de sus declaraciones a PC Magazine en 1994 en las que decía "Tendramos un ancho de banda infinito en aproximadamente una década". Y es que con la convergencia actual de tecnologías se tiende a que el cable llegue a todos los hogares, facilitando multitud de servicios como VOD, televisión interactiva, juegos, etc.

Aprovechando este entorno la red podría convertirse en un "bus" y cada terminal en el hogar, según SUN, un NC, en un procesador, con lo que toda la red se convertiría en superordenador con Java como S.O. De ahi el slogan de SUN "*The Network is the Computer*".

BIBLIOGRAFÍA

- John R. Hines, "Java: jive?". IEEE Spectrum, March 1996, p.15
- Greg Gillespie, "Whatís hot? The NC is no PC, but it could be the wave of the future in computing". The Institute, IEEE, March 1996, vol 20, num.3, p.1,4
- Arthur van Hoff, Shami Shiao, Orca Starbuck, SUN Microsystems Inc. "Hooked on Java", Addison-Wesley,1996.
- Trudy E. Bell, John A. Adam, Sue J. Lowe, "Communications", IEEE Spectrum. Technology 1996. Analysis and Forecast Issue. January 1996, p.30

PROXIMAMENTE

El próximo mes esta sección también se dedicará a Java. Se mostrará el manejo de *applets* y su inclusión en págnas Web. Asimismo, se explicará el entorno de desarrollo de Java: *JDK* y quizá en el próximo CD-ROM venga incluída una sorpresa...

CONCLUSIONES

El terreno que se pisa es inestable. Parece claro que algo va a ocurrir, pero no lo que no está claro es cuál va a ser la evolución del panorama en los próximos meses. Los planes que SUN tiene para Java presentan una nueva filosofía por lo que el autor reconoce que aún no ha obtenido ninguna conclusión. Quizá lo mejor sea esperar con los ojos bien abiertos y ver qué sucede. Mientras tanto, se anima a lector a que saque sus conclusiones y exprese su opinión mediante el correo del lector o poniéndose en contacto con el autor mediante:

E-mail internet: echeva@dit.upm.es
E-mail Compuserve: 100646,2456
WWW: http://highland.dit.upm.es:8000

Las referencias corresponden todas a artículos del mismo autor publicados en Sólo Programadores, según:

[Echeva-1], "Tecnología del WWW". n.15
[Echeva-2], "El lenguaje HTML (I)". n 16
[Echeva-3], "El lenguaje HTML (II)". n.17
[Echeva-4], "CGI: Envío de información a un servidor". n.21
[Echeva-5], "Third International WWW Conference". n10.
[Echeva-6], "La interfaz CGI: Primer Contacto". n20
[Echeva-7], "Bases de datos de red". n.22.

IMPLEMENTACIÓN DEL JUEGO DEL AHORCADO

José C. Remiro



A partir de esta entrega se presentarán una serie de programas en donde se aplicarán todos los conceptos que se han desarrollado en anteriores artículos. El lenguaje seleccionado para implementar los programas será Turbo Pascal (Borland) versión 7.0 (es posible que funcione con la versión 5.0 y posteriores), debido a que se trata de un lenguaje fácil de aprender y comprender, es fuertemente estructurado y además es similar al seudocódigo utilizado en esta sección, además, el compilador seleccionado y su entorno de programación son muy buenos.

Para aquellos lectores que estén interesados en esta sección y no dispongan de dicho compilador, o bien hayan seleccionado otro lenguaje para desarrollar sus programas se espera que esto no sea un inconveniente, pues a lo largo del artículo se explicará con detalle como se ha construido el programa y el medio de almacenamiento de datos que acompaña a la revista lo incluirá, además, actualmente todos los lenguajes imperativos presentan instrucciones similares y traducir un programa de un lenguaje a otro no presenta casi ningún problema.

Se comenzará desarrollando un programa para implementar el conocido juego del ahorcado. Se trata de seleccionar una palabra que el jugador tendrá que adivinar, para ello seleccionará un conjunto de letras que de encontrarse en la palabra, aparecerán colocadas en los lugares correspondientes, si las letras seleccionadas no se encuentran en la palabra se producirá un error pasándose a dibujar la parte correspondiente a la figura de un ahorcado, si el jugador es capaz de adivinar la palabra antes de cometer 7 fallos habrá ganado el juego, de no ser así el jugador habrá perdido.

En este artículo y sucesivos, se pasará de la teoría a la práctica, desarrollando programas sencillos que intenten aplicar los conocimientos expuestos en los anteriores entregas de esta sección. Se presenta en este caso la implementación de un juego muy sencillo y por todos conocido: el ahorcado.

ESTUDIO PREVIO DEL PROBLEMA

Se comenzará diseñando la pantalla que se presentará al jugador. En este programa se ha dividido la pantalla en cuatro zonas:

La zona de selección de letras, donde el usuario seleccionará la letra deseada.

La zona de mensajes, donde aparecerán mensajes acerca de cómo jugar y del resultado final del juego.

La zona de visualización del ahorcado, a través del dibujo se indicará el número de errores cometidos.

La zona de visualización de la palabra a adivinar, donde se indicará la longitud de la palabra e irán apareciendo cada una de las letras que el usuario, tras seleccionarlas, halla adivinado.

Para presentar la pantalla se puede elegir entre el modo texto (24 filas x 80 columnas), donde solamente se pueden visualizar caracteres y el modo gráfico, donde se puede manejar la pantalla pixel a pixel, además de poder aprovechar el entorno gráfico que Borland ofrece. Debido a la naturaleza del programa a desarrollar se ha elegido el modo gráfico. En el cuadro 1, se muestra la pantalla junto con las diferentes zonas.

El primer problema que se presenta para realizar este programa, es como validar las entradas que puede realizar el usuario. La primera decisión de diseño ha consistido en restringir la teclas válidas, así, el usuario solamente podrá utilizar las teclas de cursor y las teclas <Intro> y <Esc>. De esta forma, podrá abandonar el programa pulsando la

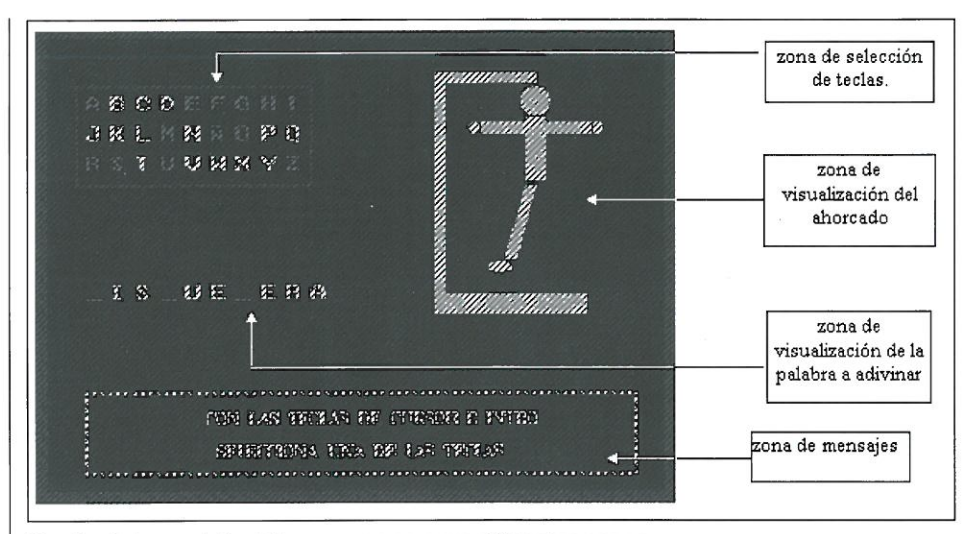
tecla <Esc> y podrá seleccionar una letra moviéndose por la zona de visualización de letras, mediante las teclas de cursor y finalmente pulsando <Intro>.

El segundo paso que se ha seguido durante el diseño es simplificar un poco más el problema. Evidentemente, el jugador desearía que el programa tuviera la posibilidad de presentar una nueva palabra tras haber terminado de jugar con la actual, pero para diseñar el programa es más fácil hacerlo para la presentación de una única palabra y posteriormente incluir esta solución en un bucle que compruebe la condición de salida que habrá introducido el usuario. Una vez diseñada esta parte del algoritmo se pasará a extender la solución, por tanto habrá que tener en cuenta el problema de la inicialización de variables que ha de producirse cada vez que se presente al usuario una nueva palabra. En el cuadro 2 se presenta un algoritmo muy general que muestra las acciones a realizar por el programa, mientras que en el cuadro 3 aparece el cuerpo del programa en Turbo Pascal.

El siguiente paso, consiste en decidir como representará la información que utiliza el programa. Evidentemente, el programa ha de tomar las palabras de algún lugar y que mejor que un archivo de acceso directo para hacerlo. El problema que se presenta, es seleccionar cada vez una palabra diferente de las anteriores y que además, cada vez que el usuario ejecute el programa las presente en orden diferente. La solución que se ha dado, es seleccionar la posición que ocupa una palabra dentro del archivo de forma aleatoria utilizando para ello la función *random* y el procedimiento *randomize.* La anterior función devuelve un número aleatorio entre cero y el número que se le ha pasado como parámetro (el parámetro con el que se llamará a *random* será el número de registros del archivo) y el anterior procedimiento selecciona una nueva semilla para generar dicho número aleatorio.

Para representar la palabra que el usuario tiene que adivinar no basta con una cadena de caracteres, pues habrá que distinguir entre las letras de la cadena que han sido acertadas por el usuario y aquellas que no ha adivinado. Para este fin, se ha definido la estructura de datos *cadena*, ésta consta de dos campos, uno de tipo string para almacenar la palabra a adivinar y un array de tipo lógico de igual longitud que el string, de tal forma que si en la posición i tiene almacenado el valor *true* representa que la letra de posición i dentro de la palabra ha sido acertada, en caso contrario dicha posición contendrá el valor *false.*

Similar estrategia se ha seguido para representar la estructura de datos que representará a las letras que podrá seleccionar el usuario. Una letra puede



Cuadro 1. La pantalla del programa con sus diferentes zonas.

```
programa ahorcado

inicializar_variables
inicializar_archivo
inicializar_pantalla
mientras no_finalizar
    presentar_pantalla_inicial
    seleccionar_palabra
    mientras no palabra_adivinada y no_finalizar
            seleccionar_letra
            comprobra letra_en_palabra
            presentar_letra_en_pantalla
    fin_mientras
    Si palabra_acertada
       entonces
         presentar_mensaje_enhorabuena
       en otro caso
         presentar_mensaje_no_adivinada
    Fin Si
fin_mientras
restaurar_pantalla
Fin Programa
```

Cuadro 2. Algoritmo general de las acciones del programa.

estar en tres estados diferentes: seleccionada (el usuario ya la ha elegido y no podrá volver a seleccionarla), en proceso de selección (el usuario ha marcado la letra pero aún no ha pulsado la tecla <Intro> para confirmar su selección) y actual (indica que la selección ha sido confirmada). El tipo de datos *abcda*, es un array de 27 posiciones (una por letra) que contiene una letra y un valor de tipo lógico para indicar si la letra ha sido seleccionada o no. Para indicar la letra actual, y puesto que un instante dado solamente puede haber una letra en estado actual, se utilizará una variable que contendrá la posición del array donde se encuentra, en el cuadro 4 se presentan las estructuras de datos utilizadas.

Teniendo en cuenta lo indicado anteriormente, cuando el usuario selecciona una letra pueden ocurrir dos casos generales, bien que la letra ya haya sido seleccionada (se indicará al usuario mediante el cambio de color de la letra) en cuyo caso se producirá un error en la entrada de datos (el programa lo indicará emitiendo un sonido) o bien que la letra no halla sido seleccionada con anterioridad, en este caso el proceso a realizar es evidente, se buscará la letra seleccionada en la palabra a adivinar asignando, en caso de encontrarla en la palabra, el valor *true* a la posición del campo *aciertos* que ocupa la misma posición que la letra acertada. Si por el contrario, no se encontraba dicha letra en la palabra el jugador habrá errado y por tanto habrá que visualizar la parte correspondiente del muñeco que indica los fallos que hasta el momento ha tenido el usuario.

```
begin   {del programa principal}
 tray_archivo:= 'd:\ahorcado\palabras.lst';
 if existe_archivo (tray_archivo)
   then
    begin
     assign (arch_palabras, tray_archivo);
     reset(arch_palabras);
     codigo_tecla:=0;
     ocultar_cursor;
     while (codigo_tecla <> tecla_escape)
           do
            begin
             obten_palabra (arch_palabras, pal_adivinar.palabra);
             for i:= 1 to 12
                 do
                  pal_adivinar.aciertos[i]:= false;
             inicia_lista_letras (lista_letras);
             modo_grafico;
             inicio;
             seleccion_teclas (codigo_tecla, pal_adivinar);
             closegraph;
            end;
     close (arch_palabras);
    end
   else
    writeln('programa finalizado. el archivo ', tray_archivo, ' no
     encontrado');
 end.
```

Cuadro 3. Cuerpo del programa principal en Turbo Pascal.

En este programa se ha dividido la pantalla en cuatro zonas

Pero las acciones que han de realizarse si se ha adivinado una letra no están aún completas, pues habrá que indicar al usuario que efectivamente ha adivinado una letra de la palabra, así, habrá que visualizar todas las ocurrencias de dicha letra en la zona de visualización de la palabra a adivinar. Pero es posible además que la palabra haya sido hallada en su totalidad, en cuyo caso habrá que visualizar en la zona de mensajes, un aviso de felicitación para el usuario y habrá terminado de jugar con la palabra actual, por lo que además se deberá visualizar algún mensaje que así lo indique.

También, está incompleta la descripción de las acciones a realizar cuando se falla en la selección de una letra, pues habrá que decidir si este fallo es el séptimo, en cuyo caso de nuevo el usuario terminará de jugar con la palabra actual, por tanto, se deben visualizar las letras que aún le quedaban por acertar, un mensaje de condolencia y un aviso para que decida si desea continuar jugando o no.

Tras terminar con la palabra actual y si el usuario ha decido continuar con el juego habrá que presentar la pantalla inicial, inicializando todas las variables para comenzar con una nueva palabra, por el contrario si el usuario decide abandonar el programa habrá que restaurar la pantalla a modo texto y salir al DOS.

Además de controlar los errores que puedan producirse por una entrada errónea, habrá que controlar los errores que puedan producirse al no encontrar el programa algunos archivos que necesite o bien que la tarjeta gráfica a utilizar no sea la correcta. Así, antes de visualizar la pantalla inicial, el programa deberá comprobar la existencia del archivo de palabras y de los archivos correspondientes a la interfase gráfica y si el tipo de tarjeta gráfica utilizada es VGA. De no ser así se emitirá un mensaje adecuado y el programa finalizará.

DESCRIPCIÓN DEL PROGRAMA

El programa comienza asignando a la variable *tray_archivo*, la trayectoria del archivo que contiene las palabras que propondrá el programa al usuario y comprobará que éste existe, de no ser así finalizará el programa emitiendo un mensaje de error indicando al usuario que no se ha encontrado el archivo. Este proceso es realizado por la función *existe_archivo*, utiliza la directiva del compilador *{$I-}* para desactivar los errores que emite el compilador ante el error de una operación de entrada salida y poder así utilizar un tratamiento particular desde el programa.



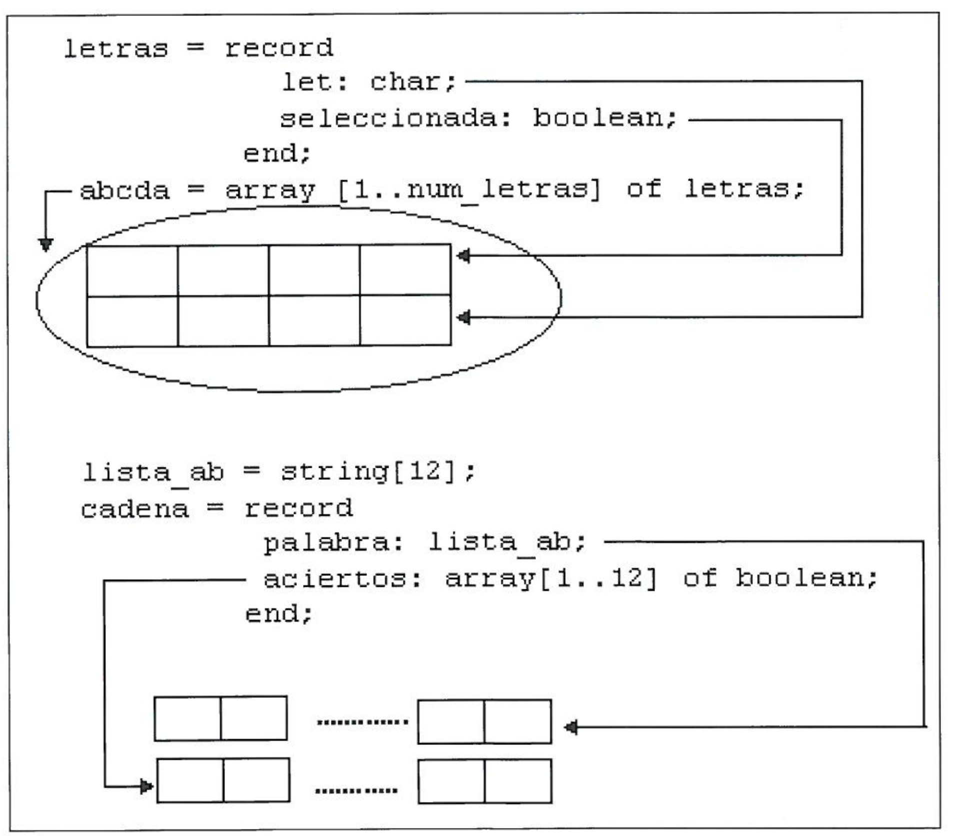
Si se ha encontrado el archivo de palabras se procede a su apertura en modo sólo lectura, se asigna a la variable *codigo_tecla* el valor cero para poder entrar en el bucle que comprueba si el usuario desea seguir jugando y se oculta el cursor.

Tras comprobar que el archivo de palabras se ha encontrado, el programa entra en el bucle que realiza la selección de una palabra del archivo, inicializando el array de tipo lógico, que hace corresponder a cada posición de las letras de la palabra el valor *false* (letra no encontrada). Además, se visualizan las letras del abecedario con todos los valores del campo *seleccionado* a *true*, lo que indicará que todas las letras son seleccionables por el usuario. Esta acción la realiza el procedimiento *inicia_lista_letras*.

Tras estas inicializaciones, el programa establece el modo gráfico a través del procedimiento *modo_grafico*, en el se detecta el tipo de tarjeta gráfica que utiliza el sistema. El programa solamente está preparado para trabajar con tarjetas VGA, si hubiese algún problema a la hora de detectar la tarjeta, este procedimiento abortaría el programa a través de la orden *halt(1)*, emitiendo un diagnostico de la causa que provocó el fallo.

Tras establecer el modo gráfico, se le presenta al usuario la pantalla inicial, que consta de la zona de selección de letras, donde aparecerán todas las letras del abecedario, en color no seleccionado y con la letra activa *A*, en la parte inferior de la pantalla se situará la zona de mensajes, visualizando un mensaje escueto sobre lo que debe hacer el usuario en cada momento, y por último, aparece indicado mediante el carácter subrayado, el número de caracteres que tiene la palabra a adivinar.

Una vez que se han realizado todas estas acciones pasa a ejecutarse el procedimiento más complejo del programa, *selección_teclas*, que admite como parámetros el código de la tecla pulsada y la palabra a adivinar, el primero de estos parámetros sirve para devolver el código de la tecla que el usuario ha pulsado, cuando se le hace la pregunta desde dentro del procedimiento de si desea o no continuar jugando, comunicándoselo así al bucle en el que está inmersa su llamada. Posteriormente se comentará más detenidamente como realiza sus funciones este procedimiento.



Cuadro 4. Estructuras de datos utilizados en el programa.

Tras finalizar el tratamiento con una palabra, se pasa a cerrar el modo gráfico, esto permite borrar toda la pantalla en caso de que el usuario desee continuar el juego o bien restaurar la pantalla a modo texto en caso de que desee finalizar.

EL TRATAMIENTO DE UNA PALABRA

En el procedimiento *seleccion_teclas* se realiza el tratamiento de una palabra, dispone de dos parámetros por variable *tecla_pulsada* que realiza la función de devolver la tecla pulsada tras responder el usuario si desea o no continuar el juego y *ad_pal* que permite modificar la estructura donde se encuentra la palabra a adivinar.

En el cuerpo del anterior procedimiento, primero se establece el valor de las diferentes teclas que serán válidas y el valor de la variable *final_palabra* a *false*, variable que indicará si el juego ha terminado o no. Tras este proceso de inicialización se produce el tratamiento de la palabra, entrando en un bucle que finalizará, bien cuando el usuario no desee continuar jugando (pulsando la tecla <Esc>), o bien cuando el tratamiento de la palabra halla finalizado (si ha sido acertada o si el usuario ha cometido 7 errores al tratar de adivinarla).

Cada vez que se acepte una tecla válida (teclas de cursor, <ESC> o <Intro>), se procederá a realizar la correspondiente acción. Así, si la tecla pulsada es de cursor, se procede a llamar al procedimiento *iluminar_letra*. Este procedimiento acepta dos parámetros: la letra que actualmente está iluminada y el código de la tecla pulsada, al ejecutarse el procedimiento se iluminará la letra que corresponda, que dependerá tanto de la letra actualmente iluminada (con color diferente que el resto de teclas) como de la tecla pulsada, obsérvese en el cuadro 5 que el cuerpo del procedimiento contiene una sentencia *case* para determinar la posición de la letra que se iluminará.

Si la tecla pulsada por el usuario es <Intro>, querrá decir que el usuario ha seleccionado la letra actual (aquella cuyo orden se encuentra almacenado en la variable *orden_letra*), habrá que comprobar en primer lugar si el usuario la había seleccionado con anterioridad (es decir, el campo *seleccionada* correspondiente a la letra tiene valor *false*) si no es

```
procedure iluminar_letra (codigo_tecla:
integer; var orden_letra: integer);
 var
  nv_orden_letra: integer;
  posicionx_carac, posiciony_carac:
        integer;
 procedure mover_arriba (var ord_let:
        integer);
  begin
   if ord_let > 9
     then
      ord_let:= ord_let - 9
     else
      ord_let:= ord_let + 18;
  end;
 procedure mover_abajo (var ord_let:
        integer);
  begin
   if ord_let > 18
     then
      ord_let:= ord_let - 18
     else
      ord_let:= ord_let + 9;
  end;
 procedure mover_derecha (var ord_let:
        integer);
  begin
   if ord_let = 27
     then
      ord_let:= 1
     else
      ord_let:= ord_let + 1;
  end;
procedure mover_izqda (var ord_let:
integer);
begin
     then
      ord_let:= 27
     else
      ord_let:= ord_let - 1;
  end;
 begin {de iluminar letra}
  nv_orden_letra:= orden_letra;
  case codigo_tecla of
      72: mover_arriba
(nv_orden_letra);
      75: mover_izqda
(nv_orden_letra);
      77: mover_derecha
(nv_orden_letra);
      80: mover_abajo
(nv_orden_letra);
  end;
  calcula_pos_letra (orden_letra,
posicionx_carac, posiciony_carac);
  moveto (posicionx_carac,
posiciony_carac);
  if lista_letras
[orden_letra].seleccionada
     then
      setcolor(color_sel)
     else
      setcolor(color_nosel);
outtext(lista_letras[orden_letra].let);
  calcula_pos_letra (nv_orden_letra,
posicionx_carac, posiciony_carac);
  setcolor(color_actual);
  moveto (posicionx_carac,
posiciony_carac);
outtext(lista_letras[nv_orden_letra].let
);
orden_letra:= nv_orden_letra;
 end;{de iluminar_letra}
```

Cuadro 5. Cuerpo del proceso iluminar letra.

así, se producirá un error emitiendo el programa un sonido de error, en otro caso habrá que establecer el campo *seleccionado* de la letra correspondiente a *true* y comprobar las ocurrencias de la letra en la palabra a adivinar, estas acciones las realiza el procedimiento *buscar_caracter*.

Este último procedimiento admite como parámetros la posición de la letra seleccionada en el array *lista_letras*, junto a éste, la palabra a adivinar y un variable de tipo lógico que será *true* si en la palabra hay al menos una ocurrencia de la letra seleccionada, además, si hay alguna coincidencia se aprovecha para visualizar todas ellas sustituyendo la letra por el carácter subrayado que aparece en la pantalla (este carácter indica que hay letras que aún no han sido adivinadas).

Tras finalizar el procedimiento *buscar_caracter*, hay una llamada al procedimiento *comprobar_final*, éste admite dos parámetros de tipo lógico, el primero le indicará si se ha acertado una letra (*true*) o no, el segundo se pondrá a *true* si se ha terminado de jugar con la palabra actual. Los otros dos parámetros son el número de errores que ha cometido el usuario hasta ahora y la palabra a adivinar. Las acciones que realiza este procedimiento son las siguientes: comprueba en el caso de haberse adivinado una letra, si todas las letras de la palabra han sido adivinadas, si este es el caso, entonces visualizará un mensaje de enhorabuena. Si no se ha adivinado una letra de la palabra se incrementa en uno el parámetro que representa el número de errores cometidos por el usuario, dibujando la parte del muñeco correspondiente al error (se corresponde con la llamada a *dibuja_ahorcado(n_err)*) y comprueba que el número de errores es siete, si esto es así, se emitirá el correspondiente mensaje.

Una vez finalizado el tratamiento correspondiente a la entrada de una letra, se procede a llamar a la función *devuelve_tecla*, admite un parámetro de tipo lógico, que indicará a la función si el tratamiento de la palabra ha terminado o no. Si ha terminado el tratamiento de la palabra, la función emite un mensaje para indicar al usuario si desea continuar o por el contrario salir del programa, puesto que la pregunta es de cierta importancia hay una parte de código que se dedica a vaciar el buffer del teclado (utilizando la interrupción 16) para que el usuario vea primero el mensaje antes de realizar ninguna acción, el código se muestra en el cuadro 6. Por el contrario, si el usuario no ha terminado aún con la palabra a adivinar, se procede a aceptar teclas, seleccionando sólo aquellas que sean válidas.

CONDICIONES PARA LA COMPILACIÓN DEL PROGRAMA

Si se desea compilar el programa y ejecutarlo se debe tener en cuenta algunas consideraciones:

Son necesarias las unidades *DOS*, *Graph* y *CRT*, estas se encuentran normalmente en el directorio *UNITS* dentro del directorio donde se tenga instalado el compilador (normalmente el directorio *TP*). De no ser así, en la opción *Options*, *Directories* especificar donde se encuentran las unidades.

El programa espera encontrar el archivo donde se encuentran las palabras en la trayectoria *D:\ahorcado\palabras.lst*, habrá que modificar la primera línea del cuerpo del programa antes de compilarlo si esta trayectoria se ha variado.

Son necesarios los archivos del Interface Gráfico de Borland, situado en el directorio *BGI*, que se encuentra normalmente situado dentro del directorio donde se ha instalado Turbo Pascal.

El programa, para ser ejecutado necesita de una tarjeta VGA.

Si se siguen las anteriores recomendaciones podrá ser compilado y ejecutado con éxito el programa. Se incluye un archivo de 50 palabras aproximadamente, si se desea ampliarlo se deberá implementar una rutina que defina un archivo con registros de tipo string de 12 caracteres, introducir en él todas las palabras del archivo que acompaña al del programa y a continuación seguir introduciendo las palabras desde el teclado. Posteriormente se borrará el archivo inicial y se renombrará el nuevo archivo con el nombre del anterior.

De todas formas este programa admite muchas mejoras como la comprobación de la existencia de todos los archivos necesarios para la ejecución del programa, integración de la función de añadir palabras al archivo, inclusión de records y estadísticas, inclusión de pistas para adivinar las palabras, inclusión de niveles, mejora de la pantalla etc., ampliar el programa con estas y otras funciones podría ser un buen método para aprender a programar o ampliar los conceptos de programación del lector.



RETROCEDA QUIEN SE RECONOZCA

Ya de bebé, Miguel era de esos niños que adhieren los ojos a ciertas cosas, que con la cabecita quieta y la mirada grave anclan la vista durante minutos estáticos e interminables sobre un solo objeto de su habitación. Hasta cuando miraba a su madre, el pequeño Miguel lo hacía con un ceño de incomprensión lejana.

Lo cierto es que Miguel nunca reconfortó a sus seres cercanos exhibiendo un solo sentimiento de pertenencia a nada ni nadie, aunque practicara muestras de un afecto disciplinado y mimético hacia aquéllos a los que habría sido desagradecido no amar. Miguel creció como un niño tan despierto como ausente, extraño en todas partes pero con una buena disposición a cumplir con las expectativas de sus padres. A los tres años leía de corrido.

Cumplidos los siete, mientras los vecinos de su edad chillaban como monitos y se destrozaban las rodillas en el polvo de la explanada a la hora del pan con chocolate, Miguel, un chiquillo pálido de piernas flacas, prefería devorar libros y tebeos y pegar la mejilla en el espejo de su habitación. Lo primero le llevaba a una alienación absoluta. Se transportaba al universo del libro, ya fueran los feudos submarinos del Capitán Nemo o las azoteas desde donde Spiderman dominaba la noche de la ciudad, y lo hacía con un autoabandono tal, que cuando un libro o tebeo se le acababa, apenas tomaba una bocanada del aire real de su habitación antes de sumergirse en la atmósfera de otro libro que le consolara de la despedida del primero.

En los momentos en que pegaba el rostro al espejo del dormitorio, se transportaba mentalmente al otro dormitorio, al mundo especular, tan imaginario pero tan real, y a través de la ventana de ese dormitorio encerrojaba la mirada y la mente en la otra calle, la calle al revés que la de verdad. Por alguna razón, ese dormitorio invertido parecía más amplio y menos restringido, y el mundo al que conducía su ventana más espacioso, más lleno de posibilidades y libre de trámites y barreras. Su madre no dejaba de decirle que tomara el aire, que saliera más. Miguel ya salía. De hecho, pasaba días y noches fuera. Sólo que su cuerpo se lo dejaba en su cuarto, como el submarinista que deja una boya en la superficie durante la inmersión. Miguel siguió creciendo en la cuneta de la realidad, siempre poco problemático y siempre inquietante. Empezó a trabajar como programador de videojuegos. Dos o tres chicas intentaron encontrar el camino de su corazón, pero lo que les impidió el paso no fueron corazas de hierro o muros de piedra, sino una niebla fría y espesa.

Era un programador excelente. Con algoritmos sencillos construía mundos llenos de riqueza y complejidad, y todos veían en sus juegos un poder especial para transportar a quien jugaba, para sacarle del asiento y colarle en la pantalla. Hay que decir que Miguel cada vez se salía más de sí mismo cuando hacía su trabajo, y cuando jugaba a sus propios juegos era cuando se hacía más intemporal e intangible. Cada vez pasaba más tiempo delante del ordenador. Sin variar la postura ni la posición de los ojos, movía sólo las falanges de los dedos, tecleando rápida y silenciosamente, comunicándose cada vez más con su máquina, cada vez más incapaz de hacerlo con una realidad que le parecía crecientemente bidimensional y ausente de sorpresas. "Los átomos no me interesan", decía. "Los bits son lo que cuenta". Defendía su argumento ante sus compañeros, en las pocas ocasiones en que rompía su silencio. Este mundo de átomos no tenía nada que decirle. Los átomos son errantes, infieles, y por ello completamente sustituibles. Abandonan por completo nuestro cuerpo para ser sustituidos por nuevos átomos cada siete años. Lo que los mantiene en formación no es otra cosa que nuestra memoria, el conjunto de bits que configura lo que recordamos consciente e inconscientemente sobre nosotros mismos.

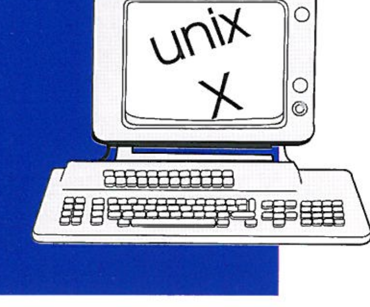
Nunca encontró placer en las cosas ligadas a la prosaica materia: una buena comida, el cuidado de una planta o el dulce ejercicio del amor. Al contrario, cada vez se volcaba más en sus algoritmos, se emocionaba ante detalles sutiles de sus rutinas que pasaban inadvertidos al resto del mundo. Esto precipitó su caída, su progresiva abdicación del reino de la realidad. Pasaba la mayor parte del tiempo metido en casa. Ya sólo se comunicaba por correo electrónico y por faxes mandados desde el ordenador, en los que la firma era siempre la misma imagen digitalizada y almacenada en el disco duro. Su rendimiento como programador aumentó aún más. Producía cada vez más algoritmos y más complejos. Sus juegos cada vez se parecían más a él. Una chica que le quiso intentó sacarle de su soledad, pero sólo consiguió hacerse daño. Acabó jugando a sus juegos como única manera de tocarle.

Cortó todo contacto físico con las personas con quienes se relacionaba. Su trabajo estaba siempre listo a tiempo, transferido por correo electrónico al servidor de la compañía. Así pasaron varios meses, hasta que un día dejaron de llegar los mensajes.

En su casa sólo encontraron sus objetos y su ropa. Sus átomos, celosos de su espíritu, le habían impuesto una reválida que él no supo aprobar. Uno a uno, dejaron de relevarse y le fueron abandonando hasta que sus bits, el recuerdo que tenía de sí mismo, tuvieron que emigrar a sus algoritmos y no encontraron el camino de vuelta. Se fue volviendo cada vez mas translúcido hasta que alcanzó la total transparencia. Desapareció, transferido a sus juegos, como dicen los árabes que ocurre cuando las cámaras fotográficas te roban el alma.

BASES DE DATOS EN RED

Fernando J. Echevarrieta



En anteriores artículos se han realizado aplicaciones cliente-servidor en las que el cliente conocía de antemano la dirección del servicio. En el presente, se facilita el conjunto necesario de herramientas para que el servidor pueda atender en cualquier puerto de cualquier máquina de Internet permitiendo su localización mediante la consulta de bases de datos de red

Como se ha venido haciendo patente a lo largo de los últimos meses, el IPC (InterProcess Communication) de UNIX se basa en el modelo cliente-servidor. Una primera solución para la comunicación entre procesos aparecía con el uso de pipes y el mecanismo de Ken Thompson; sin embargo, como se estudió en su momento, resultaba insuficiente para la comunicación entre distintas máquinas ya que exigía que los procesos tuvieran un antepasado común, por lo que debían formar parte del mismo sistema operativo. La variante de Berkeley, la estudiada en la serie, se basaba en un iterfaz de sockets, como abstracción de puntos finales de comunicación y la de SYSTEM V, se basaba en otros mecanismos no estudiados como mensajes, memoria compartida, semáforos, etc.

Hasta el momento, en los ejemplos vistos en la programación de comunicaciones mediante sockets (*internetworking*), las direcciones de las máquinas y puertos se encontraban "cableadas" en el código. Sin embargo, algunos lectores habrán reparado que en el caso de los ejemplos de RPCs, que también se sustentan sobre sockets, era posible indicar una máquina destino refiriéndose a ella por su nombre. También, hasta el momento, se ha centrado más el tema de los artículos en el mecanismo de comunicación entre los procesos. En el presente se introducirán las bases de datos de red necesarias para la gestión de nombres y direcciones de redes, máquinas y servicios que permitirán que el software de comunicaciones que estamos realizando sea capaz de encontrar cualquier programa servidor en Internet.

DIRECCIONAMIENTO IP

En todo sistema cliente-servidor es necesario establecer un contacto entre las dos partes, por lo que debe existir un sistema de identificación y localización de los servicios. Para ello habrá, en primer lugar, que localizar la máquina destino a partiendo de una dirección. En el caso de dos máquinas conectadas a un mismo medio físico, se emplean las denominadas direcciones de *nivel 2* o direcciones físicas. Una red como Internet, proporciona de forma lógica una abstracción de lo que es un enlace entre dos máquinas, permitiendo la comunicación de ordenadores que se encuentran conectados a distintos medios físicos en distintas redes. Las direcciones de máquinas en redes son lo que se denomina direcciones de *nivel 3* y se trata de direcciones lógicas. Y protocolos de nivel 3 o protocolos *de red* serán los encargados de realizar una comunicación entre dos máquinas de la red se encuentren donde se encuentren. En el caso de Internet, el protocolo correspondiente es *IP* (*Internet Protocol*) y las direcciones reciben, por tanto, el nombre de *direcciones IP*.

Una dirección IP viene dada por 32 bits y para su uso cómodo por el ser humano, se suele facilitar en formato ASCII agrupada por cadenas de bytes separados por puntos. Por ejemplo: 138.4.22.25 correspondería a 10001010000001000001011000011001.

LOCALIZACIÓN DE SERVICIOS

Contrariamente a lo que a primera vista pueda parecer, el problema del establecimiento de una comunicación entre máquinas va mucho más alla que el de

la simple localización de la máquina destino. Así, no basta con indicar que se desea realizar una comunicación con la máquina *138.4.22.25*, ya que en esta máquina existen multitud de servicios, como *ftp*, *http*, *telnet*, *echo* y un largo etcétera. Del mismo modo que para localizar a una persona no basta saber el portal sino también piso y puerta, es necesario utilizar una dirección más específica para localizar cada servicio concreto. Es lo que se denomina dirección de *nivel 4*. En el mundo de Internet, estas direcciones se contruyen mediante la dirección IP de la máquina, ampliada con un número denominado *puerto*. Los protocolos encargados de realizar esta comunicación entre dos entidades finales se denominan protocolos de nivel 4 o protocolos *de transporte* y, como se ha visto en ocasiones anteriores, Internet dispone de dos: *UDP*, no fiable, basado en datagramas y *TCP*, fiable, orientado a conexión y que, junto con IP da nombre a toda la arquitectura de protocolos de Internet: *TCP/IP*.

NOMBRES Y DIRECCIONES

Aun con la representación de grupos de bytes separados por puntos, el sistema no es exactamente cómodo para el ser humano si tiene que recordar las direcciones IP de todas las máquinas con las que trabaja en su red. Por otra parte, si una máquina se cambia de red, deberá cambiar su dirección IP. Así, desde el principio se pensó en asignar nombres a las máquinas que tuvieran una equivalencia con su dirección IP de tal forma que aunque cambiara esta dirección se pudiera conservar el nombre. De esta forma, surgieron las primeras bases de datos para conversión de direcciones IP a nombres y viceversa. La mayoría de las bases de datos con información de red en UNIX se encuentra en ficheros situados en el directorio */etc*. Para la resolución de nombres y máquinas se pueden consultar estos ficheros u otras bases de datos presentes en la red como *NIS* (*Network Information Service*) de SUN o *DNS* (*Domain Name System*) que se tratarán en futuros artículos.

Las correspondencias entre nombres y direcciones IP de los hosts accesibles al sistema aparecen en el fichero */etc/hosts*

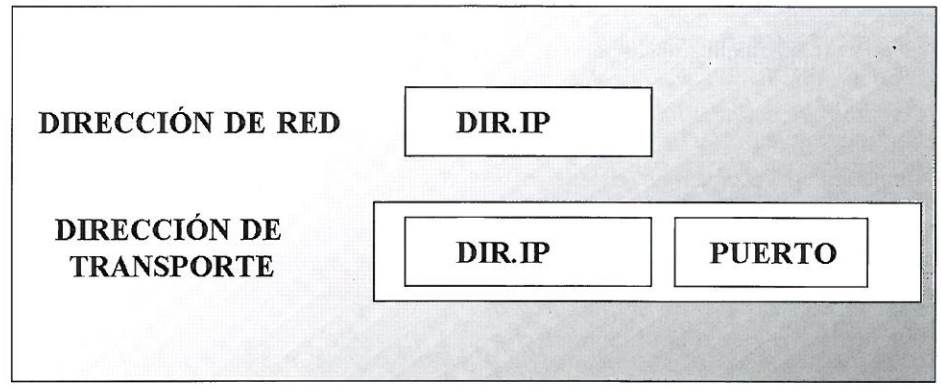


figura 1: Direcciones de red y transporte.

que consta de tres campos principales: *dirección nombre_oficial alias...*

Los alias son otros nombres por los que se puede conocer a una misma máquina además del nombre oficial. Un ejemplo de fichero */etc/hosts* prodría ser:

127.0.0.1 localhost
138.4.2.10 sanson
138.4.2.11 cuco
138.4.1.129 cuco-gw
138.4.2.1 idit-atm

En él se pueden observar dos características de las direcciones IP. En primer lugar se puede observar la dirección *127.0.0.1* que, como se comentó en su día, siempre corresponde al un bucle local. Según el estándar la dirección de red *127* no puede aparecer nunca en la red. El host 1 se asigna a un dispositivo virtual de bucle local con el fin de utilizarse en pruebas que es, al fin y al cabo, lo que se ha estado haciendo en los capítulos anteriores de la serie.

La segunda y más importante de las características es que las direcciones IP no identifican máquinas sino conexiones de red y cada una de ellas puede estar asociada con un nombre diferente. Así pues, las direcciones *138.4.2.11* y *138.4.1.129* corresponden a dos conexiones de red de la misma máquina. Es decir, esta máquina se encuentra conectada a dos subredes, en una recibe el nombre de *cuco* y en la otra el de *cuco-gw*. Esto puede crear cierta confusión al ser humano, que siente tendencia a identificar cada máquina con un único nombre. Pero lo que puede causar aún más perplejidad es que según llame a una máquina de una forma u otra, su servicio puede comportarse de forma más o menos eficiente o, incluso, "no comportarse". Para entenderlo mejor, imagine el lector la configuración de la figura 2.

Supóngase que la conexión de red número 1 de la máquina A recibe el nombre de *soloprog* y la conexión 2 de la misma máquina, el de *solop*. Si la máquina B establece una conexión con *solop*, es probable que ésta sea menos eficiente que si la establece con *soloprog*. Aunque se trata de la misma máquina y sea el mismo servidor el que vaya a atender la petición, en el primer caso, la latencia de la comunicación será mayor ya que el tráfico deberá realizar un "salto" a través de C. Por otra parte, mientras *soloprog* y la conexión 1 de la máquina B se encuentran en la misma red, no ocurre lo mismo con *solop* y la conexión 2 de la máquina B. La información deberá competir por el ancho de banda con el tráfico de una red, primero, y de otra, después. La situación puede llegar al límite si, en determinado momento, la máquina C cae o, por algun motivo se abre una cualquiera de las dos redes 2 ó 3. En cualquiera de los tres casos, la conexión con *solop* se hará imposible. ¡Poco importará que las máquinas A y B se encuentren directamente conectadas!

Una dirección IP es una dirección de nivel 3 con un tamaño de 32 bits

CONSULTA A BASES DE DATOS DE RED

Al desarrollar software de comunicaciones será necesario obtener direcciones IP dado un nombre, números de puerto

dado un servicio, obtener direcciones de red, etc. Para ello, se dispone de una serie de rutinas de consulta a las bases de datos adecuadas. Para la utilización de estas rutinas será necesario siempre incluir en los programas el fichero de cabeceras *netdb.h*. Con el objeto de ilustrar las rutinas disponibles así como realizar ciertas puntualizaciones sobre ellas se ha realizado un simple programa interactivo que realiza algunas consultas y cuya fuente se puede observar en el listado 1.

BÚSQUEDA DE MÁQUINAS

Para la consulta de máquinas se dispone de dos funciones de librería básicas: *gethostbyname(nombre_host)*, que realiza una búsqueda por nombre y *gethostbyaddr(dir,long_dir,tipo_dir)*, que realiza una búsqueda por dirección de red.

Ambas devuelven un puntero a una estructura denominada *hostent* definida de la forma:

```
struct hostent {
  char *h_name;
  char **h_aliases;
  int h_addrtype;
  int h_length;
  char **h_addr_list;
};
#define h_addr h_addr_list[0]
```

en la cual se obtendrá información proporcionada por el servidor de nombres *named*, que se estudiará en el futuro, o extraída del fichero */etc/hosts* o el *NIS*. En esta estructura:

h_name contiene una cadena con el nombre oficial de la máquina terminado en '\0'.

h_aliases es una lista de alias terminados en '\0'.

h_addrtype indica el tipo de dirección (generalmente *AF_INET*)

h_length indica la longitud en bytes de la dirección (4 para *AF_INET*).

h_addr_list devuelve la lista de direcciones IP asociadas a la máquina de las cuales, la principal se puede obtener también de *h_addr* por compatibilidad con versiones anteriores.

Como se recordará de [Echeva-1] para convertir una dirección IP del formato de caracteres separados por puntos a su representación como entero largo, se disponía de la rutina *inet_addr(dir)*. La conversión inversa

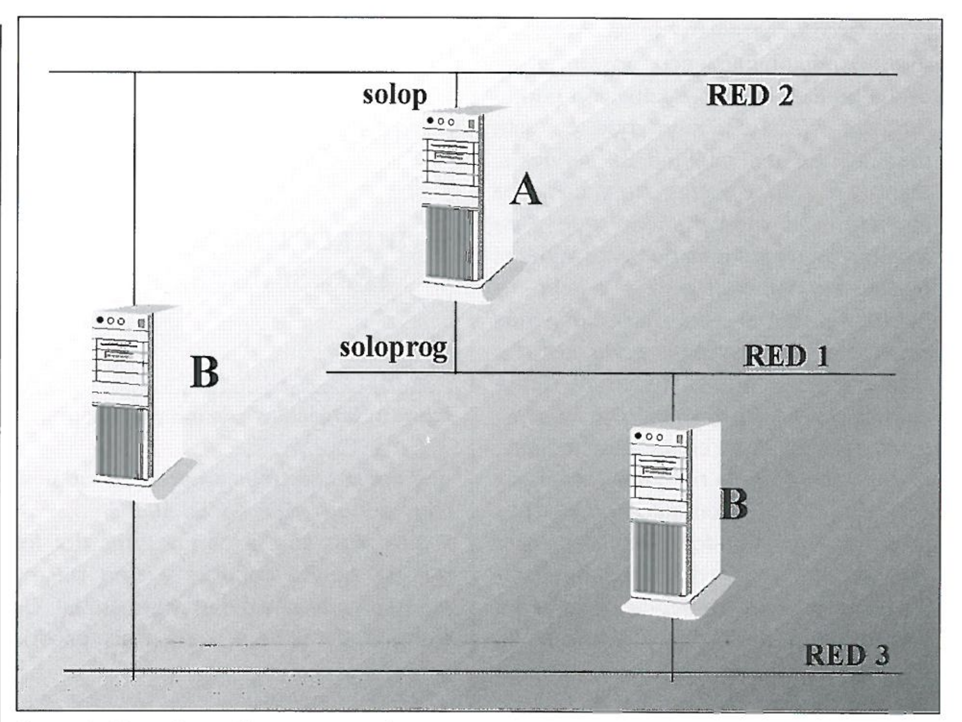


figura 2: Direcciones IP como conexiones de red.

también se puede realizar mediante la función *inet_ntoa(dir)*, donde *dir* debe ser una estructura *in_addr*:

```
struct in_addr {
  unsigned long int s_addr;
}
```

En el listado ejemplo, para realizar la búsqueda del nombre dada una dirección, se pide ésta al usuario, quien la introduce a través del teclado (suele ser usual que se facilite como parámetro). Por ello es necesario realizar una primera conversión:

```
sin.s_addr=inet_addr(dir);
```

la copia mediante *bcopy* y NUNCA con *strcpy*. Esto se debe a que aunque la dirección se devuelve mediante un puntero a char, se trata simplemente de un puntero a una zona de memoria (quizá habría sido más acertado definirlo como puntero a void) y el contenido no está formado por caracteres sino que se trata de un número que podría contener un byte 0 que una rutina de impresión o copia de cadenas consideraría fin de la cadena. La longitud de los datos a copiar se corresponderá con la longitud de la dirección facilitada en *h_length*:

Para acceder a un servicio no es suficiente con saber en qué máquina se encuentra

Además, dado que la dirección se le pasa a la rutina como un puntero a char, será necesario realizar un casting de la misma:

```
host=gethostbyaddr((void *)&sin.s_addr,4,AF_INET)
```

Cuando se desee recuperar una dirección, ya sea para realizar una conexión copiándola al campo correspondiende de la estructura *sockaddr_in* que se le pasará a un *connect* [Echeva-2] o, como en el ejemplo del listado, para mostrarla, será necesario realizar

```
bcopy(host->h_addr,(void *)&sin.s_addr,host->h_length);
```

En el ejemplo, en que la información se muestra por pantalla, se realiza de nuevo una conversión a ASCII separado por puntos mediante *inet_ntoa*:

```
printf("Direccion: %s\n",inet_ntoa(sin));
```

BÚSQUEDA DE SERVICIOS

Al igual que ocurre con los nombres de máquina, es más fácil conocer un servicio por su nombre, por ejemplo *telnet*, que por su número de puerto, *23*. Del



mismo modo, es posible colocar un servicio atendiendo en cualquier puerto. Así pues, surge de nuevo la necesidad de realizar conversiones nombre-puerto. Como ya se comentó en artículos anteriores, únicamente los servidores pertenecientes a root podrán utilizar los puertos hasta el 1024 por razones de seguridad [Echeva-3]. Aunque la localización de puertos no suele problema para el usuario final, ya que los programas clientes se dirigen por defecto a los puertos convenientes, sí será un problema para el programador de estos clientes: en este caso, el lector.

En el caso de una comunicación cliente-servidor, lo lógico es que el servicio se identifique mediante un nombre conocido y público. El cliente, en cambio, podrá disponer de un puerto aleatorio que le asigne dinámicamente el sistema al ser creado.

Los servicios conocidos para una máquina se registran en el fichero */etc/services* que tiene en cada línea una entrada que responde a la forma:
nombre_oficial puerto/protocolo alias...
en el que *protocolo* especifica el protocolo de transporte TCP o UDP por el que es accesible el servicio en ese puerto.

Para su consulta desde programa se dispone de las rutinas:
getservbyname(nombre,protocolo), que realiza una búsqueda por el nombre del servicio y
getservbyport(puerto,protocolo), que realiza una búsqueda por el puerto de atención.

En ambos casos será necesario indicar el protocolo de transporte que se desea utilizar (TCP o UDP) como se explicará más adelante con el programa *echochli*. Ambas rutinas, devuelven la información en otra estructura:

```
struct servent {
  char *s_name;
  char **s_aliases;
  int s_port;
  char *s_proto;
}
```

en la que *s_name* y *s_aliases* son el nombre y lista de alias correspondientes a un servicio; *s_port* el puerto por el que se encuentra accesible y *s_proto*, el protocolo de transporte empleado.

LISTADO 1

```
LISTADO 1
/* consulta.c - Consulta de bases de datos de red
*/
/* Fernando J. Echevarrieta (echeva@dit.upm.es)
*/

#include <stdio.h>
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

char nom_dir();

main()
{
  char opcion='0';

  while (opcion=='0') {
    printf("1.Busca MAQUINA\n");
    printf("2.Busca SERVICIO\n");
    printf("3.Busca PROTOCOLO\n");
    printf("Elige [1-3]:");

    opcion=getc(stdin);
    while (getc(stdin)!='\n');
    switch (opcion) {
      case '1' : maquina(); break;
      case '2' : servicio(); break;
      case '3' : protocolo(); break;
      default  : printf("Opcion no valida\n\n");
    }
  } /* while */
}

maquina()
{
  struct hostent *host;
  char nombre[255], dir[17];
  int i=0, exito=0;
  struct in_addr sin;

  switch (nom_dir()) {
    case '1' : printf("Nombre : ");
 scanf("%s",nombre);
      if (host=gethostbyname(nombre)) exito=1;
 break;
    case '2' : printf("Direccion IP: ");
 scanf("%s",dir);
              sin.s_addr=inet_addr(dir);
 if                (host=gethostbyaddr((void
*)&sin.s_addr,4,AF_INET)) exito=1;
 break;
  }

  if (exito) {
b c o p y ( h o s t - > h _ a d d r , ( v o i d
*)&sin.s_addr,host->h_length);
    printf("DATOS DE LA MAQUINA:\n");
    printf("Nombre oficial: %s\n",host->h_name);
    while (host->h_aliases[i] != NULL)
              printf("Aliases  [%d]       :
%s\n",i+1,host->h_aliases[i++]);
    printf("Tipo direccion: %d\n",host->h_addrty-
pe);
    printf("Longitud dir  : %d\n",host->h_length);
    printf("Direccion     : %s\n",inet_ntoa(sin));
    i=1;
    while (host->h_addr_list[i] != NULL) {
b c o p y ( h o s t - > h _ a d d r _ l i s t [ i ] , ( v o i d
*)&sin.s_addr,host->h_length);
printf("Direccion [%d] : %s\n",i++,inet_ntoa(sin));
    }
  }
  else printf("No encuentro esa maquina\n");
}

servicio()
{
  struct servent *serv;
  char nombre[255], proto[255];
  int puerto, i=0, exito=0;

  printf("Protocolo (TCP/UDP): ");
  scanf("%s",proto);
  while (getc(stdin)!='\n');
  switch (nom_dir()) {
    case '1' : printf("Nombre del servicio: ");
          scanf("%s",nombre);
          if (serv=getservbyname(nombre,proto))
exito=1;
          break;
    case '2' : printf("Puerto: ");
                   scanf("%d",&puerto);
if      (serv=getservbyport(htons(puerto),proto))
exito=1;
          break;
  }

  if (exito) {
    printf("DATOS DEL SERVICIO:\n");
    printf("Nombre oficial: %s\n",serv->s_name);
    while (serv->s_aliases[i] != NULL)
      printf("Alias [%d]      : %s\n",i,serv->s_alia-
ses[i++]);
printf("Puerto       : %d\n",ntohs(serv->s_port));
    printf("Protocolo    : %s\n",serv->s_proto);
  }
  else printf("No encuentro ese servicio\n");
}

protocolo()
{
  struct protoent *proto;
  char nombre[256];
  int exito, numero, i=0;

  switch (nom_dir()) {
    case '1' : printf("Nombre del protocolo: ");
          scanf("%s",nombre);
            if (proto=getprotobyname(nombre))
exito=1;
          break;
    case '2' : printf("Numero: ");
                   scanf("%d",&numero);
          if (proto=getprotobynumber(numero))
exito=1;
          break;
  }

  if (exito) {
    printf("DATOS DEL PROTOCOLO:\n");
    printf("Nombre oficial: %s\n",proto->p_name);
    while (proto->p_aliases[i] != NULL)
      printf("Alias [%d]     : %s\n",i,proto->p_alia-
ses[i++]);
    printf("Numero       : %d\n",proto->p_proto);
  }
  else printf("No encuentro el protocolo\n");
}

char nom_dir()
{
  char op='0';

  while (op=='0') {
    printf("1.Por nombre\n");
    printf("2.Por numero\n");
    printf("Elige [1-2]:");

    op=getc(stdin);
    while (getc(stdin)!='\n');
    if ((op<'1') || (op>'2')) {
          printf("Opcion no valida\n");
      op='0';
    }
    else return op;
  }
}
```

En este caso, la única aclaración a realizar es que el número de puerto (entero corto) se facilita y se devuelve en orden de bytes de red [Echeva-2], por lo que en el listado 1 ha sido necesario convertir mediante *htons* (entero corto) el puerto tecleado por el usuario al formato externo para realizar la llamada:

serv=getservbyport(htons(puerto),proto)

y de nuevo al interno, mediante *ntohs*, para mostrar la información:

printf("Puerto: %d\n",ntohs(serv->s_port))

BÚSQUEDA DE REDES Y PROTOCOLOS

Menos frecuentes son los casos en que es necesario identificar un determinado protocolo o una red. No obstante se dispone también de herramientas para ello.

Los protocolos conocidos por una máquina se registran en el fichero */etc/protocols* que responde a la forma:

nombre_oficial num_protocolo alias ...

En el listado 2 puede observarse un caso típico.

En este fichero, como en cualquiera de los otros mencionados en este artículo, cualquier texto que aparezca tras un símbolo hash '#' será considerado un comentario.

Análogamente a los casos anteriores, se dispone de dos rutinas para la identificación de protocolos:

getprotobyname(nombre)
getprotobynumber(numero)

que realizan una búsqueda por nombre o número de protocolo devolviendo una estructura del tipo:

```
struct protoent {
  char *p_name;
  char **p_aliases;
  int p_proto;
}
```

en la que, como es de adivinar, *p_name* devuelve el nombre del protocolo; *p_aliases*, una lista de alias y *p_proto*, el número de protocolo. El significado de estos números está relacionado con la estructura de las PDUs [Echeva-2] de los protocolos a los que corresponden por lo que, de momento, no se profundizará más en el tema.

La identificación de redes es el último caso que se tratará en este artículo siendo formalmente idéntico a los anteriores. El fichero de registro es */etc/networks* cuyas líneas tienen una estructura

nombre_oficial numero_red

precisamente al revés que el */etc/hosts*.

Para obtener información sobre las direcciones de red se dispone de las rutinas

getnetbyname(nombre)
getnetbynumber(número)

que realizan una búsqueda por nombre y dirección de la red y devuelven un puntero a una estructura *netent*:

```
struct netent {
  char *n_name;
  char **n_aliases;
  int n_addrtype;
  long n_net;
}
```

en la que *n_name* y *n_aliases* contienen el nombre y una lista de alias de la red respectivamente, *n_addrtype*, el tipo de dirección y *n_net*, la dirección.

EL CLIENTE ECHOCLI

Como complemento al artículo se ha realizado un simple cliente TCP del servicio de *echo* que implementan todas las máquinas UNIX para la realización de pruebas. Como se puede apreciar en el listado 3, este servicio se encuentra implementado tanto para TCP como para UDP. En el primer caso, el servidor se limita a devolver por la conexión que le realiza un cliente todo lo que reciba por ésta mientras que en el segundo hace lo propio devolviendo datagramas al cliente.

LISTADO 2

```
#
# @(#)protocols 1.9 90/01/03 SMI
#
# Internet (IP) protocols
# This file is never consulted when the NIS are running
#
ip      0    IP     # internet protocol, pseudo protocol number
icmp    1    ICMP   # internet control message protocol
igmp    2    IGMP   # internet group multicast protocol
ggp     3    GGP    # gateway-gateway protocol
tcp     6    TCP    # transmission control protocol
pup     12   PUP    # PARC universal packet protocol
udp     17   UDP    # user datagram protocol
```

El software cliente deberá ser capaz de traducir nombres de servicios como www al número de puerto correspondiente: 80

El listado debe servir como ejemplo al lector para la localización de direcciones y servicios en sus propios programas. Para ello, se parte de un cliente como los realizados en artículos anteriores al que se le pasa como único parámetro *argv[1]* el nombre o la dirección IP de la máquina servidora. El cliente realiza una búsqueda por nombre y si tiene éxito, la copia al campo correspondiente de una estructura *sockaddr_in* para su posterior utilización en un *connect*. Como se indicaba en un punto anterior, habrá que tener especial cuidado al copiar la dirección y realizar el casting correspondiente:

*bcopy(host->h_addr,(void *)&TCPsock.sin_addr.s_addr,host->h_length);*

Si la función *gethostbyname* falla el programa considera que se le ha facilitado una dirección IP en formato ASCII con separación de puntos y trata de convertirla a un entero largo mediante *inet_addr*. Si esta operación también falla, deduce que el parámetro era un nombre no encontrado, lo notifica y termina.

Del mismo modo realiza una búsqueda de un servicio de *echo* que se sustente sobre TCP ya que se trata de un cliente orientado a conexión. Si tiene éxito en su búsqueda de máquina y ser-



LISTADO 3

```
/* echocli.c - Cliente TCP de echo            */
/* Fernando J. Echevarrieta (echeva@dit.upm.es)
*/

#include <stdio.h>
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>

#define MSG "ECO ECO... CONCENTRADO
ESTOYYYYY"

main(int argc, char **argv)
{
   char buf[256];
   int s;

   struct sockaddr_in TCPsock;
   struct servent *serv;
   struct hostent *host;

   TCPsock.sin_family=AF_INET;

   /* Nivel 3 (maquina): RED (IP)*/
   if (host=gethostbyname(argv[1]))
                      bcopy(host->h_addr,(void
*)&TCPsock.sin_addr.s_addr,host->h_length);
   else
 if  ((TCPsock.sin_addr.s_addr=inet_addr(argv[1]))
== INADDR_NONE)
       {
         printf("No encuentro el host: %s\n",argv[1]);
         exit(1);
       }

   /* Nivel 4 (puerto): TRANSPORTE (TCP/UDP) */
   if (serv=getservbyname("echo","tcp"))
      TCPsock.sin_port=serv->s_port;
   else {
      printf("No encuentro el servicio de echo\n");
      exit(1);
   }

   s=socket(AF_INET,SOCK_STREAM,0);
   if (connect(s,(struct sockaddr *)&TCPsock,size-
of(TCPsock))<0) {;
        printf("No puedo conectar con el servidor:
%s\n", sys_errlist[errno]);
      exit(1);
   }
   write(s,MSG,sizeof(MSG));
   read (s,buf,sizeof(MSG));
   close(s);

   printf("He enviado  el mensaje: %s\n",MSG);
   printf("He recibido el mensaje: %s\n",buf);
}
```

vicio realiza la conexión e informa de su resultado.

OTRAS RUTINAS DE BÚSQUEDA

Ademas de las rutinas tratadas, existe un grupo de otras tres para cada base de datos a consultar que reciben el nombre de *set<b_datos>ent*, *get<b_datos>ent* y *end<b_datos>ent* donde *b_datos* puede ser *host*, *serv*, *proto* o *net*. Esta serie de rutinas facilitan un acceso secuencial a las bases de datos de red proporcionando todo su contenido. Cada conjunto sigue un patrón de comportamiento común:

1. Establecimiento de una "conexión" con la base de datos
2. Lectura de entrada de una en una (una por cada llamada)
3. Cierre de la "conexión".

Así, por ejemplo, para la consulta de hosts se definen las rutinas *gethostent*, *sethostent* y *endhostent* cuyo comportamiento es el siguiente:

sethostent(stayopen)
en caso de que exista un servidor de nombres (DNS) pedirá el uso de un socket conectado (TCP) para realizar a través del mismo peticiones al servidor. Si el argumento *stayopen* es distinto de cero, se mantendrá abierta la conexión tras cada llamada a *gethostbyname* o *gethostbynumber*. En caso de que no exista un servidor de nombres, abrirá la base de datos de la máquina local.

gethostent()
lee (de forma secuencial) la siguiente entrada de la base de datos y devuelve un puntero a una estructura *hostent* como las dos rutinas básicas de búsqueda.

endhostent()
cierra la conexión con el servidor de nombres o el fichero de la base de datos según proceda.

CONCLUSIÓN

En el presente artículo se han expuesto algunos principios básicos para la administración de redes basadas en UNIX. Asimismo se han proporcionado las herramientas necesarias para que el software de comunicaciones que el lector realice sea capaz de conectarse con cualquier máquina y servicio de internet aun sin conocer la dirección y el puerto a priori. Se propone como ejercicio la modificación del cliente de buscaminas que se publicó en el número 18 o los clientes aparecidos en el número 21 para que sean capaces de conectarse a un servidor que se encuentre en una máquina distinta. Otra posible modificación seria el registro de los servicios en el */etc/services* para que los clientes puedan conectarse con los servidores sin conocer a priori el número de puerto.

CONTACTAR CON EL AUTOR

Para cualquier duda, comentario, sugerencia o crítica se anima al lector a que se ponga en contacto con el autor mediante:
E-mail Internet: echeva@dit.upm.es
E-mail compuserve: 100646,2456
WWW:
http://highland.dit.upm.es:8000

TABLA DE PROTOTIPOS DE LAS FUNCIONES

```
#include <netdb.h>
struct protoent *getprotoent(void);
struct protoent *getprotobyname(const char *name);
struct protoent *getprotobynumber(int proto);
void setprotoent(int stayopen);
void endprotoent(void);
struct hostent *gethostbyname(char *name);
struct hostent *gethostbyaddr(char *addr, int len, int type);
struct hostent *gethostent();
sethostent(int stayopen);
struct servent *getservent(void);
struct servent *getservbyname(const char *name, const char *proto);
struct servent *getservbyport(int port, const char *proto);
void setservent(int stayopen);
void endservent(void);
struct netent *getnetent();
struct netent *getnetbyname(char *name);
struct netent *getnetbyaddr(long net, int type);
void setnetent(int stayopen);
void endnetent();
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
char *inet_ntoa(struct in_addr in);
unsigned long int inet_addr(const char *cp);
```

REFERENCIAS

Todas las referencias corresponden a otros artículos del autor publicados en Sólo Programadores según:
[Echeva-1] "Arquitecturas Cliente-Servidor" n.17
[Echeva-2] "Arquitecturas Cliente-Servidor" n.18
[Echeva-3] "Instalacion de servidores WWW" n.19

TÉCNICA DE LOS PUNTOS FUNCIÓN

José María Peco

El tema que se presenta este mes se olvida un poco de la máquina, a fin de presentar una técnica muy extendida en las grandes instalaciones para la evaluación de proyectos y que puede ser aplicada no sólo en los grandes sistemas, sino en proyectos de cualquier entorno.

Como es de imaginar, un gran sistema siempre va acompañado de unas grandes instalaciones en las que poder realizar los desarrollos. En estos desarrollos interviene mucho personal, por lo que, para poder planificar el trabajo de todo este personal, se necesitan unas herramientas que permitan determinar el coste de cada proyecto antes de iniciar su desarrollo. El método que se presenta este mes es el más utilizado en Estados Unidos para determinar la magnitud de cada proyecto, no debiéndose confundir con las herramientas propias de la gestión y dirección del proyecto, planificación de recursos, etc.

La ventaja más importante de este método estriba en su independencia del hardware, software o plataforma de desarrollo, ya que, a diferencia del método usado hasta la fecha para la evaluación de proyectos, no basa su cálculo en la estimación de un número de líneas de código (*LOC: Lines Of Code*), lo cual no permite comparar aplicaciones escritas en diferentes lenguajes, tales como COBOL, Natural o C++, y, además, depende de la agilidad o experiencia de los programadores de la instalación.

MÉTODO 2

El método ideado en 1979 por Allan Albrecht, empleado de IBM, es un método de evaluación de proyectos que permite medir la productividad de una instalación desde el punto de vista del usuario, no del programador. Para ello, se basa en el cálculo de unos valores, denominados *puntos función* (de ahí el nombre dado a esta técnica), aunque también se le conoce como *Método 2*.

Esta técnica permite evaluar el esfuerzo a aplicar a un nuevo desarrollo con el rigor adecuado, analizando la relación costo/beneficio con la aproximación deseada en una etapa temprana del diseño, exactamente cuando se cierra el diseño lógico de la aplicación y, por consiguiente, antes de iniciar la fase de desarrollo.

Entre las características más importantes, cabe destacar las siguientes:

- Este método proporciona una medida empírica, es decir, se ajusta para cada instalación en base a muchos desarrollos, no en base a una medida teórica.
- Contempla el sistema como una caja negra que simplemente realiza las funciones definidas por el usuario.
- Proporciona una unidad de medida lineal, es decir, si un sistema se evalúa en X Puntos Función, significa que su desarrollo requiere la mitad de esfuerzo que otro proyecto evaluado en 2X Puntos Función.
- Su cálculo se puede realizar en fases tempranas del proyecto, concretamente al final de la etapa del Diseño Conceptual. De esta forma, se obtiene una evaluación aproximada, pues no se encuentran definidos todavía todos los elementos que intervienen en el sistema y una evaluación definitiva al final de la etapa del Diseño Lógico.
- Otra de las características importantes de este método consiste en que no mide el esfuerzo invertido en las modificaciones del diseño del proyecto debidas a indefiniciones o definiciones incompletas en el momento de inicio del desarrollo. Por consiguiente, debe ponerse especial interés en tener perfectamente definido el sistema cuando se efectúa la evaluación.
- Por último, éste es un método muy sencillo, pues sólo requiere para su

cálculo efectuar unas pocas operaciones completando un pequeño formulario.

DESCRIPCIÓN DEL MÉTODO

El cálculo de los Puntos Función se realiza siguiendo los siguientes pasos:

1.- Identificar los elementos que intervienen en la aplicación.
2.- Evaluar la magnitud de la aplicación en base a la complejidad de los elementos identificados en el punto anterior, obteniendo unos Puntos Función no ajustados.
3.- Evaluar el grado de influencia de la instalación.
4.- Establecer la evaluación final del proyecto en Puntos Función.

PASO 1: IDENTIFICAR ELEMENTOS

En este primer paso se identifican todos los elementos de la aplicación con relevancia, para lo cual se registran en una matriz, como la de la figura 1, todos los elementos que intervienen en el proyecto, de acuerdo con los siguientes tipos:

- *Ficheros Internos*: Son aquellos ficheros con datos propios de la aplicación, o con información de control, que serán usados y mantenidos por el sistema. Se debe entender por fichero una agrupación homogénea de información, por lo que, si el continente es el mismo pero el contenido varía en función de un tipo de registro, cada tipo será considerado como un fichero interno lógico distinto.
- *Ficheros externos*: Son aquellos ficheros del sistema que no son actualizados por la aplicación pero que sí son accedidos en consulta. Cabe realizar la misma observación que en el punto anterior sobre los tipos de registro.
- *Entradas*: Objetos definidos para facilitar la entrada de datos.
- *Salidas*: Objetos definidos para soportar la información de salida.
- *Consultas*: Objetos definidos para realizar consultas sin actualización.

Ficheros Internos (Cuadro 1)		Numero de campos		
		< 5	5 a 15	> 15
Relaciones	< 2	BAJA	BAJA	MEDIA
	= 2	BAJA	MEDIA	ALTA
	> 2	MEDIA	ALTA	ALTA

Ficheros Externos (Cuadro 2)		Numero de campos		
		< 20	20 a 50	> 50
Relaciones	1	BAJA	BAJA	MEDIA
	2-5	BAJA	MEDIA	ALTA
	> 5	MEDIA	ALTA	ALTA

Entradas (Cuadro 3)		Numero de campos		
		< 5	5 a 15	> 15
Ficheros accedidos	< 2	BAJA	BAJA	MEDIA
	= 2	BAJA	MEDIA	ALTA
	> 2	MEDIA	ALTA	ALTA

Salidas (Cuadro 4)		Numero de campos		
		< 6	6 a 19	> 19
Ficheros accedidos	< 2	BAJA	BAJA	MEDIA
	= 2	BAJA	MEDIA	ALTA
	> 2	MEDIA	ALTA	ALTA

Cuadros para el cálculo de la complejidad de los elementos que intervienen como parámetros de la evaluación.

PASO 2: EVALUACIÓN EN BASE A COMPLEJIDAD

Una vez identificados cada uno de los elementos que intervienen como parámetros de la evaluación, se procede a establecer el grado de complejidad de cada uno de ellos, en base al número de campos que contienen, y a las relaciones en las que participa, de acuerdo con los siguientes cuadros:

- *Ficheros Internos*: El cuadro 1 establece el grado de complejidad en base al número de campos elementales de cada fichero interno (o tabla de la aplicación), y el número de relaciones en las que interviene en el modelo de datos de la aplicación.
- *Ficheros Externos*: El cuadro 2 establece el grado de complejidad en base al número de campos elementales consultados por la aplicación de los ficheros de uso general de la instalación, y el número de relaciones en las que interviene en el modelo de datos de la aplicación.
- *Entradas*: El cuadro 3 establece el grado de complejidad en base al número de campos elementales actualizados en cada proceso de alta, y el número de ficheros accedidos para poder realizarla.
- *Salidas*: El cuadro 4 establece el grado de complejidad en base al número de campos elementales presentados en cada salida, y el número de ficheros accedidos para poder obtenerla.
- *Consultas*: Para establecer el grado de complejidad de cada consulta individual se debe considerar que cada una cuenta con una entrada de datos para la captura de los parámetros de la consulta, y una salida con el resultado de la

El método se basa en el cálculo de unos valores, denominados Puntos Función

consulta. Por esta razón, se utilizarán los cuadros 3 y 4 para evaluar la complejidad de cada consulta.

La calificación obtenida por cada elemento, pasa a completar la matriz obtenida en el paso 1, tal y como muestra la figura 1.

El documento así obtenido, sirve para calcular los Puntos Función no ajustados, de acuerdo con el formulario de la figura 2.

TIPO	NOMBRE	CAMPOS	RELACIONES	CALIFICACIÓN
F_Interno	Entidad 1	n1	r1	Baja
	Entidad 2	n2	r2	Media
	..	...	...	...
	Entidad n	n3	rn	Alta
F_Externo	Entidad m1	m1	rx1	Baja
	Entidad m2	m2	rx2	Baja
	..	...	...	...
	Entidad mn	m3	rxn	Media
Entradas	Entrada 1			
...	e1	re1		
Salidas	Salida 1			
...	s1	rs1		
Consultas	Entrada c1			
Salida c1				
Entrada c2				
Salida c2				
...				

Figura 1. Matriz con los elementos de la aplicación con relevancia.

PASO 3: DETERMINACIÓN COEFICIENTE CORRECTOR

A continuación, se procede a determinar el factor de ajuste debido a las características de la instalación.

Cada uno de los 14 puntos que siguen, se deben puntuar de 0 a 5, y la suma de todos ellos define el *grado de influencia* (*GI*) de la instalación sobre el proyecto.

1.- *Comunicación de Datos*: En este punto se califica en qué medida los datos de la aplicación se actualizan y consultan por teleproceso.

2.- *Funciones distribuidas*: Mide la existencia de los procesos o datos distribuidos.

3.- *Rendimiento*: Determina en qué medida el sistema debe tener en cuenta tiempos de respuesta, optimización de procesos, etc.

4.- *Uso de la configuración*: Califica el grado en el que la aplicación va a ser utilizada en momentos de carga elevada de la máquina, o si va a ser muy utilizada.

5.- *Volumen de transacciones*: Este punto determina en qué medida influye el número de transacciones por unidad de tiempo en el diseño, instalación y mantenimiento de la aplicación.

6.- *Entrada On-Line*: Este punto evalúa en qué medida la entrada de datos se realiza de forma interactiva.

7.- *Eficiencia del usuario*: Este punto mide el grado de eficiencia del usuario de la aplicación, es decir, si el usuario tiene conocimientos de la operativa de la aplicación, si se pueden usar códigos en lugar de descripciones, etc. Esta característica influye en gran medida en el diseño conversacional de la aplicación.

8.- *Actualización On-Line*: Mide en qué medida se actualizan *On-Line* los datos.

9.- *Procesos Complejos*: Este apartado mide el grado de complejidad de la aplicación en su conjunto, determinando en qué medida existen procesos que requieren numerosas decisiones y actualizaciones para una transacción de alta o actualización y que, en consecuencia, deberán deshacerse en caso de no superar los controles establecidos en la aplicación, o bien por decisión del usuario.

10.- *Reusabilidad del código*: Este apartado establece una calificación de los procesos de la aplicación, determinando si el código de dichos procesos se escribe para ser utilizado sólo por la aplicación a medir, o si va a ser usado por otras aplicaciones de la instalación. A modo de orientación se establece el siguiente baremo para el porcentaje de código reusado por la instalación:

- más del 50% : 5
- de 40 al 50% : 4
- de 25 al 40% : 3
- de 10 al 25% : 2
- Hasta el 10% : 1
- No se usa por la instalación: 0

11.- *Facilidad de instalación*: Este punto mide el grado de facilidad de los distintos factores que intervienen en la carga inicial e instalación de la aplicación, o si se necesitan realizar procesos complejos para efectuar la carga inicial.

12.- *Facilidad de operación*: el calificador asignado a esta característica mide los factores de operatividad de la aplicación, determinando el grado de intervención manual para procedimientos de rearranque, seguridad, etc.

13.- *Instalaciones Múltiples*: Esta característica mide si la aplicación va a correr en una o varias instalaciones o máquinas.

14.- *Facilidad de cambio*: Este punto determina el grado de adaptación de la aplicación a los cambios futuros.

Este método contempla el sistema como una caja negra que realiza las funciones por el usuario

COMPLEJIDAD

Baja	Media	Alta	Total
F. Interno	___ * 7	___ * 10	___ * 15
F. Externo	___ * 5	___ * 7	___ * 10
Entradas	___ * 3	___ * 4	___ * 6
Salidas	___ * 4	___ * 5	___ * 7
Consultas	___ * 4	___ * 5	___ * 7

Total Puntos Función No Ajustados (PFNA) :

1. Comunicación de Datos
2. Funciones distribuidas
3. Rendimiento
4. Uso de la configuración
5. Volumen de transacciones
6. Entrada On-Line
7. Eficiencia del usuario
8. Actualización On-Line
9. Procesos Complejos
10. Reusabilidad del código
11. Facilidad de instalación
12. Facilidad de operación
13. Instalaciones Múltiples
14. Facilidad de cambio

Grado de Influencia : Total (GI)

FA = 0.65 + (GI/100) FA = 0.65 + (___/100) = ____

PF = PFNA * FA PF = _____ * _____ = _______

Figura 2. Cálculos de los puntos función no ajustados.

El *Grado de Influencia* (*GI*) de la instalación se obtiene sumando las puntuaciones dadas a cada una de las 14 características citadas, obteniéndose, por tanto, un valor comprendido entre 0 y 70.

PASO 4: CÁLCULO DE LOS PUNTOS FUNCIÓN AJUSTADOS

El *Factor de Ajuste* (*FA*) se obtiene mediante la siguiente fórmula:

FA = 0.65 + (GI / 100)

En consecuencia, su valor estará comprendido entre 0,65 y 1,35.

El cálculo de los *Puntos Función Ajustados* (*PFA*) correspondientes al proyecto a evaluar se obtiene multiplicando los *Puntos Función No Ajustados* (*PFNA*), obtenidos en el paso 3 por el Factor de Ajuste (*FA*), tal y como muestra el cuadro de la figura 2.

EVALUACIÓN DE LAS MODIFICACIONES AL PROYECTO

Este método también permite evaluar en Puntos Función el esfuerzo a realizar para implementar una modificación, ya sea por haber añadido nuevas funciones al diseño inicial, o bien por haber modificado o eliminado funciones ya evaluadas.

En cualquier caso, los pasos a seguir son:

Un equivalencia válida podría ser 1 PF = 3.5 hombres/día

- Evaluar los puntos función de las nuevas funciones
- Evaluar los correspondientes a las funciones modificadas (por diferencia entre las actualizadas y las viejas)
- Evaluar los correspondientes a las funciones eliminadas.

Los Puntos Función correspondientes a la modificación del proyecto y que determinan el esfuerzo a invertir en la misma se determinan por la suma de todos los puntos obtenidos en los pasos citados.

BIBLIOGRAFÍA

Artículo: *"Using FUNCTION POINTS to measure IS productivity"* publicado por Gartner Group el 5 de Agosto de 1992

CONSIDERACIONES FINALES

Tal y como se decía en la introducción, este método se ha implantado en casi todas las grandes instalaciones para evaluar los costes de un proyecto. Por tanto, una vez obtenidos los Puntos Función del nuevo proyecto se deben traducir estos valores en esfuerzos hombres/mes.

La pregunta inmediata por tanto es: *"Cada Punto Función, ¿a cuantos hombres/mes equivale?"*

Respuesta: Cada instalación debe contestar esta pregunta evaluando en Puntos Función los últimos desarrollos realizados. La razón entre los esfuerzos invertidos en dicho desarrollo, y los Puntos Función obtenidos por este método, establece el valor del Punto Función para la instalación. Un valor perfectamente válido podría ser 1 PF = 3.5 hombres/día.

UTILIDAD QUE SE ACOMPAÑA

La utilidad que acompaña este mes al artículo es una calculadora desarrollada con el lenguaje *Natural*. Permite realizar cálculos aritméticos, exponenciales y logarítmicos. Consta de 4 memorias y está sujeta a todo tipo de mejoras.

La utilidad reside en poder efectuar potencias basándose en el cálculo logarítmico.

Los fuentes se encuentran en el disco que acompaña a la revista, siendo su contenido el siguiente:

JMPCALCP.TXT: Listado fuente del programa *Natural*.

JMPCALCM.TXT: Definición del mapa invocado por el programa, y que sirve de interface a la utilidad.

EJEMPLO.TXT: Contiene 2 *Hardcopys* de la utilidad en tiempo de ejecución.



CÁLCULO DE EXPRESIONES

Daniel Navarro

En el artículo anterior se hizo una introducción al proceso de generación de código dentro de un compilador, y antes de continuar desde el punto donde se dejó, se hará un breve resumen como ya viene siendo habitual en esta serie.

Las fases de análisis (léxico, sintáctico y semántico) ya se completaron, lo cual quiere decir que el compilador ya es capaz de analizar completamente el programa fuente y guardar, en la tabla de objetos, un registro con la información de cada objeto encontrado en el programa (en Letra hay tres tipos de objetos: datos, tablas y etiquetas).

Se ha visto como un sencillo analizador de expresiones basado en la siguiente regla en notación BNF:

<expresión> ::= <operando>
{ <operador> <operando> }

(Una *<expresión>* es un *<operando>* al que le pueden seguir cualquier número de secuencias *<operador> <operando>)* puede calcular expresiones de cualquier complejidad, pero con el defecto de no contemplar operadores de diferentes prioridades, realizando todas las operaciones de izquierda a derecha (2+4*2 resulta 12). También se vio, que si se aceptaba como *<operando>* de una expresión otra expresión completa entre paréntesis, se podía forzar de esta forma la prioridad de los operadores (2+(4*2) resulta 10).

Para generar un código en el lenguaje *EML* (Enhanced Machine Language, cuadro 1) que fuera capaz de evaluar una expresión, el algoritmo empleado era como se muestra a continuación (suponiendo que la expresión analizada era 2+3):

```
an_expresion(2+3)
{
  an_operando(2)
  generar_operando(LOC 2)
  mientras_venga(<operador>){
    an_operador(+)
    an_operando(3)
    generar_operando(LOC 3)
    generar_operador(SUM)
  }
}
```

Una expresión es un operando al que le puede seguir cualquier número de secuencias de operandos

Este analizador de expresiones generaría el siguiente código: *LOC 2*, *LOC 3*, *SUM*. Las instrucciones *LOC* (*load constant*) apilan una constante y el operador *SUM* (suma) desapila dos constantes y apila su suma. Luego, tras ejecutarse dicho código quedaría, en la cima de la pila, el resultado de la expresión.

Analizar una expresión no es muy complicado, sólo requiere un poco de técnica. Generar código para calcular una expresión es inmediato, sólo requiere haber realizado correctamente el análisis de la expresión

Y para finalizar este resumen, sólo falta recordar que el modelo anterior se podía ampliar aceptando otros operandos (como variables que se apilan con *LOE <dirección de la variable>* o subexpresiones) y otros operadores (como la resta con *RES*, la división con *DIV*, etc). Por ejemplo, suponiendo que 'A' es una variable que se va a ubicar en la dirección física 298 de la máquina destino, para la expresión 2*(A+3)-1/8, se generaría el código siguiente:

LOC 2; LOE 298; LOC 3; SUM; MUL; LOC 1; RES; LOC 8; DIV;

Se puede ver el orden en el que se van realizando las operaciones, teniendo en cuenta que cada operador binario desapila dos valores, opera con ellos, y después apila su resultado (nótese como la división entre 8 afecta a todas las operaciones anteriores, y no sólo al 1). Al finalizar el cálculo, queda un solo valor apilado: el resultado de la expresión. Para apilar la variable 'A' no se utiliza la instrucción *LOC 298*, porque esta instrucción apilaría el valor '298', en cambio *LOE* apilará el dato contenido en la dirección de memoria 298.

El resultado de una expresión no depende sólo de la expresión, sino también de cómo sea interpretada

DE DERECHA A IZQUIERDA

El orden en el que se realizan las operaciones dentro de una expresión (y como consecuencia, su resultado) no depende sólo de la expresión en sí misma, sino también de cómo sea interpretada.

Cuando se diseñan las reglas sintácticas de un lenguaje se debe hacer con la generación de código ya en mente, pues, por ejemplo, según se implemente la sintaxis de una expresión podrá ser o no evaluada de ciertas formas. La sencilla sintaxis de una expresión que se acaba de mostrar es correcta, pero no permite más que evaluar expresiones interpretándolas de izquierda a derecha, según se leen, sin contemplar distintos niveles de prioridad en los operadores.

Antes de estudiar un modelo que permita operadores a distintos niveles, se estudiará otro modelo sencillo de expresión que permite evaluar las expresiones de derecha a izquierda (2*3+1 resulta 8), al contrario de como se leen.

Si se define una expresión con la siguiente regla sintáctica (en lugar de la que ya se había visto):

<expresión> ::= <operando> [<operador> <expresión>]

En un principio puede parecer que no se ha cambiado nada, sino que sólo se ha variado la forma de escribirlo pues, una expresión así definida puede ser cualquiera de las siguientes (con las mismas definiciones de *<operando>* y *<operador>* que antes se usaron):

2+4
*8*3+6/2*
2(A-3)+4*

Es decir, a nivel sintáctico, esta regla es equivalente a la anterior, pues permite construir las mismas expresiones, pero cuando se implementa dicha regla en un procedimiento, queda como sigue:

```
an_expresion( ){
 an_operando( )
 generar_operando( )
 si viene (<operador>){
  an_operador( )
  an_expresion( )
  generar_operador( )
 }
}
```

Si se sigue su lógica, aunque sea un poco complejo dado su carácter recursivo, se puede observar que el código EML que genera para una expresión como 2*3+1 es:

LOC 2; LOC 3; LOC 1; SUM; MUL;

Es decir, primero se suma 3+1 y después se multiplica por 2; se ha evaluado la expresión de derecha a izquierda.

Hay lenguajes, como 'C', que además de tener operadores de varios niveles de prioridad (primero se hacen multiplicaciones y divisiones, luego sumas y restas, etc.), tienen niveles de prioridad donde se realizan operaciones de izquierda a derecha, y otros niveles donde se realizan de derecha a izquierda.

Conocer la forma en la que en un lenguaje se evalúan las expresiones es importante, al menos si se pretende que los programas funcionen. Pero es aún más importante tener mucho cuidado cuando se diseña un nuevo lenguaje en que en éste se evalúen las expresiones de forma muy parecida (si no idéntica) a como se evalúan en otros lenguajes, pues de lo contrario, se estará dificultando mucho la migración de usuarios desde otros lenguajes hacia el nuevo.

Los operadores unarios son los de mayor prioridad, y cuando hay más de uno, tiene mayor prioridad el más cercano al operando

OPERADORES DE VARIOS NIVELES

En la gran mayoría de lenguajes de programación, una expresión como 4*3+2*3 no se evalúa de izquierda a derecha (lo que resultaría 42) ni de derecha a izquierda (que resultaría 36) si no que se calculan primero las multiplicaciones y después la suma, resultando 18.

Se va a estudiar a continuación cómo se puede crear un modelo de

expresión con cuatro operadores repartidos en dos niveles de prioridad, dos de mayor prioridad (*MUL*, *DIV*) y dos de menor (*SUM*, *RES*).

Lo que se hace es definir dentro de la *<expresión>* una parte de mayor prioridad que se denominará un *<término>*. Éste será una subexpresión que sólo puede utilizar los operadores de máxima prioridad (*MUL* y *DIV*) para operar con los *<operandos>* (que son las constantes, datos, tablas, etc.):

<término> ::= <operando> { <operador_MUL_o_DIV> <operando> }

Por esta razón, se definirá una *<expresión>* como algo que sólo puede utilizar los operadores de mínima prioridad (*SUM* y *RES*) para operar con los *<términos>*:

<expresión> ::= <término> { <operador_SUM_o_RES> <término> }

Estas dos reglas sintácticas se implementarán en los dos algoritmos correspondientes, como se muestran a continuación:

```
an_expresión( ) {
  an_término( )
  mientras_venga(+ o -) {
    an_operador( )
    an_término( )
    generar_operador( )
  }
}

an_término( ){
  an_operando( )
  generar_operando( )
  mientras_venga(* o /) {
    an_operador( )
    an_operando( )
    generar_operando( )
    generar_operador( )
  }
}
```

Cuando estos dos procedimientos analicen una expresión como 2*3, sucederá lo siguiente:

Al ser llamado *an_expresión()*, lo primero que hace es invocar a *an_término()*. Éste analiza el operando 2, genera un *LOC 2* y después, como viene un operador de su nivel (*), lo analiza, y

CUADRO 1

En el artículo del número anterior (mes de Mayo) se puede encontrar la descripción completa de las instrucciones de este código, pues aquí se muestra tan sólo un breve resumen.

LOC <constante>:	apilar constante;
LOE <dirección>:	apilar [dirección];
LOF <dirección>:	desapilar a; apilar [dirección+2*a];
STE <dirección>:	desapilar [dirección];
STF <dirección>:	desapilar a; desapilar b; [dirección+b*2]=a;
MUL:	desapilar a; desapilar b; apilar b*a;
DIV:	desapilar a; desapilar b; apilar b/a;
MOD:	desapilar a; desapilar b; apilar b%a;
SUM:	desapilar a; desapilar b; apilar b+a;
RES:	desapilar a; desapilar b; apilar b-a;
NEG:	desapilar a; apilar -a;
NOT:	desapilar a; apilar not a;
AND:	desapilar a; desapilar b; apilar b and a;
OR:	desapilar a; desapilar b; apilar b or a;
DIS:	desapilar a; desapilar b; si (b!=a) apilar CIERTO; sino apilar FALSO;
IGU:	desapilar a; desapilar b; si (b==a) apilar CIERTO; sino apilar FALSO;
MAY:	desapilar a; desapilar b; si (b>a) apilar CIERTO; sino apilar FALSO;
MAI:	desapilar a; desapilar b; si (b>=a) apilar CIERTO; sino apilar FALSO;
MEN:	desapilar a; desapilar b; si (b<a) apilar CIERTO; sino apilar FALSO;
MEI:	desapilar a; desapilar b; si (b<=a) apilar CIERTO; sino apilar FALSO;
IR <dirección>:	ir dirección;
IRF <dirección>:	desapilar a; si (not a) ir dirección;
LLA <dirección>:	llamar dirección;
RET:	retorno;

Resumen de las instrucciones del código EML.

tras él el siguiente operando, generando el *LOC 3* y después *MUL*. Al regresar a *an_expresión* se finaliza el trabajo de análisis de la expresión, ya que no viene ningún operador de su nivel.

Como se puede observar, el procedimiento *an_expresión()* no hace ningún *generar_operando()* (no genera ningún *LOC* ni *LOE*), ya que esto lo hace únicamente *an_término()*, luego, cuando se analice una expresión como 4+5 sucederá lo siguiente:

Primero es invocado *an_expresión()* que a su vez invoca a *an_término()*, éste genera el *LOC 4* y como tras él no viene ningún operador de su nivel retorna. Entonces *an_expresión()*, que sí reconoce el operador +, invoca otra vez a *an_término()* que igualmente genera un *LOC 5*, y retorna a *an_expresión()*, que, por último, genera el código de su operador: *SUM*.

Si se analiza el código de estos dos procedimientos, el resultado generado por ellos para la expresión 4*3+2*3 sería el siguiente:

LOC 4; LOC 3; MUL; LOC 2; LOC 3; MUL; SUM;

Es decir, se realizan primero las multiplicaciones (de izquierda a derecha) y después la suma, evaluando la expresión según la entienden normalmente todos los lenguajes de programación.

Dentro de este modelo se puede tener igualmente subexpresiones, que se analizan como un *<operando>* que, en lugar de ser una variable o una simple constante, son otra expresión completa (con sus respectivos niveles de operadores) encerrada entre paréntesis. De esta forma, se puede, por ejemplo, forzar la expresión anterior para que sea calculada de izquierda a derecha como sigue:

(4*3+2)*3

Hoy en día, los lenguajes de programación utilizados no tienen sólo dos niveles de prioridad dentro de los operadores, sino muchos más. En el cuadro 2 se puede observar los diferentes nive-

Para implementar un modelo de expresión con varios niveles, se definen diferentes partes dentro de la expresión

deseo suscribirme a GENTE INTERNET con la siguiente modalidad: (marque con una x).

Suscripción Normal (precio según tabla).

Suscripción Extra (suplemento de 10.000 Ptas.) En pago aplazado sumar las 10.000 Ptas. al 1er plazo.

Suscripción Profesional (suplemento de 20.000 Ptas.) En pago aplazado sumar las 20.000 Ptas. al 1er plazo.

que aquí el 1er número que desea recibir y la modalidad de pago:

úmero A RECIBIR	☐ UN PAGO AL CONTADO	☐ APLAZADO 1er	2º	3º	4º	5º
1 - 2	23.630	4.995	4.995	4.995	4.995	4.995
3	23.213	4.995	4.995	4.995	4.995	4.578
4	22.796	4.995	4.995	4.995	4.995	4.161
5	22.379	4.995	4.995	4.995	4.995	3.744
6	21.962	4.995	4.995	4.995	4.995	3.327
7	21.545	4.995	4.995	4.995	4.995	2.910
8	21.128	4.995	4.995	4.995	4.995	2.493
9	20.711	4.995	4.995	4.995	4.995	2.076

R POR LA LÍNEA DE PUNTOS. DOBLE POR LA MITAD Y CIERRE CON CINTA ADHESIVA

LICE MAYÚSCULAS PARA RELLENAR ESTA TARJETA

ıbre y Apellidos .... F. nacimiento ....

ıicilio .... C.P. ....

ad .... Provincia .... Telf.:.........................

Profesión ....

RMA DE PAGO:

cargo a mi tarjeta VISA nº. _______ _______ _______ _______ Caduca ....

iciliación bancaria

ır. del banco ....

ación ....

CÓDIGO CUENTA CORRIENTE

ENTIDAD	OFICINA	DC	Nº CUENTA

ɟo a Vd. que se sirva cargar en mi ❑ cuenta corriente ❑ libreta de ahorro

s recibo/s que le será presentado por ABETO EDITORIAL S.L.

ɔ pago de mi suscripción a GENTE INTERNET.

ɔntra reembolso del importe más gastos de envío.

ıeque a nombre de ABETO EDITORIAL S.L., que adjunto*.

ro Postal (adjunto fotocopia del resguardo)*.

Firma:

. forma de pago es exclusivamente para pagos al contado

RESPUESTA COMERCIAL
Autorización nº 14.254
B.O. de C. nº 62
del 18.07.95



TARJETA POSTAL

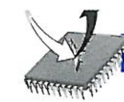
NO
NECESIT
SELLO
(A franquea
en destino)

ABETO EDITOR
APARTADO FD Nº
28080 MADRID

Rte.: ______________________________

____

____



les de prioridad que tiene el lenguaje Letra, y en el cuadro 3 la sintaxis BNF completa de una expresión en Letra el que se contemplan dichos niveles de prioridad.

OPERADORES UNARIOS

Alguien puede preguntarse porqué los operadores + y - aparecen dos veces, una con menor prioridad que los operadores multiplicativos (* / y %) y otra con mayor prioridad que ellos. Y porqué si Letra tiene 5 niveles de prioridad en la sintaxis sólo parecen contemplarse 4 (expresión, cómputo, valor y término).

La respuesta a ello se encuentra en los operadores unarios; estos son los operadores de mayor prioridad, y son: el signo positivo, el signo negativo y el operador NO (la negación de una expresión lógica, *NOT*, que en C es el símbolo '!').

Si, por ejemplo, se tiene la expresión 7*-A (siete por menos A), el signo negativo se debe aplicar a la variable A antes de realizar la multiplicación. Otro ejemplo es la expresión lógica "no encontrado y n>=-10", en ella hay dos operadores unarios: el operador NO de la variable lógica "encontrado" y el signo menos de la constante 10. estas dos operaciones deben realizarse en primer lugar, después el operador >= (mayor o igual), y por último el de menor prioridad, que es el operador Y (*and*).

El signo menos se implementa con la instrucción *NEG* de EML, que se encargará de sacar un valor de la pila, cambiarlo de signo y, por último, volver a apilarlo de nuevo; y la negación lógica se implementa con *NOT*, que saca un valor de pila y si éste equivale a CIERTO (que se representa mediante cualquier número impar) apila una constante que indique FALSO (es decir, apila un 0), y si el valor desapilado equivale a FALSO (que viene representado por cualquier número par) apila una constante que indique CIERTO (es decir, apila un 1).

Luego, si se supone que la variable "encontrado" se ubica en la dirección 333 y la variable "n" en la dirección 486, para las dos expresiones anteriores, el código que debería generarse para cada una de ellas es el siguiente, respectivamente:

CUADRO 2

Se muestran a continuación los distintos niveles de prioridad de operadores que se contemplan dentro de una expresión en Letra, los operadores del grupo superior son los de menor prioridad, y los del grupo inferior los de mayor, dentro de cada grupo se realizan las operaciones de izquierda a derecha.

NIVEL EXPRESIÓN
Y O
(and, or)

NIVEL CÓMPUTO
<> = > >= < <=
(dis, igu, may, mai, men ,mei)

NIVEL VALOR
\+ -
(sum, res)

NIVEL TÉRMINO
* / %
(mul, div, mod)

NIVEL UNARIO
NO + -
(not, neg)

Prioridad de operadores en Letra.

LOC 7; LOC A; NEG; MUL;

LOE 333; NOT; LOE 486; LOC 10; NEG; MAI; AND;

Para generar el código de esta manera, simplemente se debe definir una expresión donde un *<operando>* pueda ser, entre otras cosas, la siguiente secuencia (ver la definición completa en el cuadro 3):

<operando> ::= <OP_UNARIO>
<operando> | ...

Es decir, que un operando puede ser un operador unario seguido de, otra vez, un operando. El algoritmo recursivo que puede generarse aplicando la regla anterior puede ser el siguiente:

```
an_operando( ){
  if (viene operador unario){
    an_operador( )
    an_operando( )
    generar_operador( )
  }
  ...
}
```

Este procedimiento permite tomar un operador unario que viene justo antes de un operando y, tras analizar el propio operando (y generar su *LOC/LOE* para apilarlo), generar la instrucción que implementa la operación unaria. Este modelo no impone límite al número de operadores unarios, luego una expresión como "NO —A" generará el siguiente código:

LOC A; NEG; NEG; NOT;

Evidentemente —A equivale a poner A, pero si se hace, el compilador generará las dos instrucciones *NEG*, esto sólo se puede solucionar en una fase de optimización de código, algo que por ahora no tiene el compilador de Letra.

Deliberadamente no se ha mostrado ningún ejemplo que involucre el signo positivo. Esto es así porque éste no produce ningún efecto: las expresiones 33 y +33 son equivalentes. Luego este signo es ignorado, aunque sea analizado como los otros dos operadores unarios, no se generará ningún código para él.

Mediante las subexpresiones se permite al programador que defina el orden en el que quiere que se realicen las operaciones

SENTENCIAS DE ASIGNACIÓN

Se recuerda que, en Letra, expresión numérica y lógica es lo mismo, al menos para el compilador. Aunque para el programador una expresión lógica será la que involucre constantes lógicas (CIERTO y FALSO), variables lógicas (como "encontrado") y/o ope-

radores que produzcan resultados lógicos (como Y, O, >, >=, ...), y una expresión numérica es la que sólo involucra operandos y operadores numéricos.

Ya se puede implementar el generador de código para expresiones en Letra, aunque aún queden algunos problemas por resolver. Se ha hecho de la forma que se considera más coherente, generando código para las sentencias de asignación. Estas sentencias que acaban de ser explicadas tienen la siguiente sintáxis:

<asignación> ::= <dato> p_igual <expresión>

<dato> ::= p_identificador [p_abrir_corchete <expresión> p_cerrar_corchete]

Es decir, que se define una expresión como una referencia a un dato (una variable) o una tabla (con su correspondiente índice entre corchetes) tras la cual deberá venir el signo igual (=) y una expresión.

Generar código para una asignación supone, en primer lugar, generar código para calcular la expresión, y en segundo lugar, generar código para que se almacene el resultado en la variable a la que se hace referencia al inicio de la sentencia.

Tras calcular una expresión, el resultado queda en la pila. Para desapilarlo y ponerlo en una variable se dispone de la instrucción *STE* de EML, a la que sólo hay que pasarle la dirección de la variable. Luego, en el procedimiento de análisis de una sentencia de asignación como "a=2+-3", tras analizar el dato y el signo igual, se llamará a *an_expresión()*, que generará el código para calcular la expresión y entonces, antes de finalizar el procedimiento *an_asignación()*, se generará el *STE* para almacenar el resultado en la variable. Si la dirección de "a" fuese 1021, el código generado sería:

CUADRO 3

```
<EXPRESION> ::= <COMPUTO> { <OP_LOGICO> <COMPUTO> }

<COMPUTO> ::= <VALOR> { <OP_COMPARACION> <VALOR> }

<VALOR> ::= <TERMINO> { <OP_ADITIVO> <TERMINO> }

<TERMINO> ::= <FACTOR> { <OP_MULTIPLICATIVO> <FACTOR> }

<FACTOR> ::=
        <OP_UNARIO> <FACTOR>
        I p_abrir_parentesis <EXPRESION> p_cerrar_parentesis
        I <DATO>
        I p_cierto I p_falso
        I p_entero
        I p_literal

<OP_LOGICO> ::= p_y I p_o

<OP_COMPARACION> ::=
        p_distinto I p_igual
        I p_mayor I p_mayor_igual
        I p_menor I p_menor_igual

<OP_ADITIVO> ::= p_mas I p_menos

<OP_MULTIPLICATIVO> ::=
        p_multiplicar I p_dividir I p_modulo

<OP_UNARIO> ::= p_mas I p_menos I p_no
```

Sintáxis de una expresión en Letra

LOC 2; LOC 3; NEG; SUM; STE 1021;

ACCESO DE TABLAS: LOF Y STF

Generar código para acceder a tablas en lugar de simples datos es un poco más complicado. Supóngase que se analiza la siguiente sentencia de asignación:

*mitabla[2+3]=4*5*

En el procedimiento *an_asignación()* se analiza el identificador "mitabla", la pieza *p_abrir_corchete* y entonces se llama a *an_expresión()* para que analice el índice, con lo que se generará en primer lugar el código para calcular el índice (2+3) y dejarlo apilado (5). Tras el análisis del índice, se retorna a *an_asignación()* que se encargará de verificar que vienen la piezas *p_cerrar_corchete* y *p_igual*, y entonces se vuelve a llamar a *an_expresión()*, que analiza la expresión principal de la sentencia (4*5), generando código para calcularla y dejar su resultado apilado (20).

Es más complicado generar código para acceder a tablas que para acceder a datos

Tras calcular las dos expresiones, en la pila quedan apilados el índice y el resultado de la expresión. Ahora se tiene que generar código para poner un 20 en la posición 5 de "mitabla". Esto lo realiza directamente la instrucción *STF* de EML, que requiere como parámetro la dirección inicial de la tabla, entonces desapila primero el resultado (pues fue el último valor que se apiló), tras él desapila el valor del índice, y entonces realiza la siguiente operación (utilizando sintaxis de C):



**(mitabla+indice*2)=resultado;*

Es decir, teniendo en cuenta que cada posición de la tabla ocupa 2 bytes, pone el resultado en la dirección indicada de la tabla. Tras generar la instrucción *STF* el procedimiento *an_asignación()* retorna, pues ya ha finalizado su trabajo: analizar una sentencia de asignación y generar código para que esta se ejecute. Para la sentencia de asignación anterior se generará el siguiente código (si "mitabla" comienza en la dirección 386):

LOC 2; LOC 3; SUM; LOC 4; LOC 5; MUL; STF 386;

Como se puede ver, lo único que se debe añadir al final del procedimiento *an_asignación()* es que genere un *STE* o un *STF* en función de si la sentencia de asignación es para un dato o para una tabla, pues el código para calcular las expresiones ya se genera en *an_expresión()*.

Sólo hay un detalle que falta por ver, y es cómo se genera código cuando hay una tabla DENTRO de una expresión. Si en el programa viene una expresión como la siguiente (donde las direcciones de los objetos son a=344,b=346 y mitabla=386) *a*mitabla[b+1]*

Sucederá lo siguiente: cuando se analice el operando "a", simplemente se generará código para calcularlo con un *LOE 344*, pero cuando se analice el operando "mitabla[b+1]", éste llamará recursivamente a *an_expresión()* para calcular el índice (b+1), generando un *LOE 346; LOC 1; SUM.* Tras esto, se tiene en la pila el valor del índice, y no el contenido de "mitabla".

Nuevamente se cuenta con una instrucción de EML que solucionará el problema: se trata de *LOF;* que necesita que se le pase como parámetro la dirección inicial de la tabla, entonces desapila el índice y apila el valor que está en la posición indicada de la tabla, es decir, realiza la operación: "apilar lo que hay en mitabla[b+1]". Para completar el cálculo de la expresión anterior sólo queda multiplicar ambos operandos; el código resultante para la expresión anterior será:

LOE 344; LOE 346; LOC 1; SUM; LOF 386; MUL;

Conocer la forma en la que en un lenguaje se evalúan las expresiones es muy importante

IMPLEMENTACIÓN EN LETRA

En el disco que acompaña este mes a la revista se pueden encontrar las modificaciones que se han hecho en el compilador (*LETRA.CPP*), para implementar el generador de código para las sentencias de asignación (expresiones incluidas).

Existe una pequeño problema y es que todavía no se están asignando direcciones de memoria a los datos, es decir, no se sabe en que posición de memoria está un dato o una tabla. Se ha añadido el campo dirección dentro de los registros de la tabla de objetos, pero éste contendrá por ahora siempre 0 como dirección, así que todos los objetos parecerán estar ubicados en dicha dirección (las instrucciones *LOE,LOF,STE* y *STF* tendrán todas 0 como parámetro).

Aún no se genera código Intel 80x86, sólo se genera en un fichero denominado *EML.LST* el listado con las instrucciones EML correspondientes a cada sentencia de asignación que aparezca en el programa.Existen varias funciones menores del generador de código (*iniciar_generador()* que crea el fichero "EML.LST", *finalizar_generador()* que lo cierra y *separar_eml()* que se usa para separar dentro del listado las diferentes sentencias de asignación). Pero la función principal del generador de código es "generar (instrucción, parámetro)". A ésta se le pasa el código de una instrucción EML y, en caso de que lo requiera, su parámetro; la función genera el código para dicha instrucción (por ahora sólo la escribe en el fichero EML.LST).

Existe una estructura denominada *t_eml* que es el tipo que se emplea para contener información sobre todas las instrucciones del lenguaje EML. Por ahora sólo se especifica para cada instrucción su nombre y su número de parámetros (0 o 1), más adelante se especificará, por ejemplo, el código 80x86 equivalente a dicha instrucción.

Existe algún pequeño detalle más, pero para verlo es mejor remitirse al código del compilador. En él sólo se tiene que buscar las llamadas a la función *generar()* para localizar los puntos en los que se está generando código, y así establecer la relación con lo que se ha explicado.

El signo más es ignorado, aunque sea analizado como los otros dos operandos unarios, no se generará ningún código para él

PRÓXIMO NÚMERO

Puede parecer que aún queda mucho camino por recorrer, pero eso no es cierto. De hecho, ya se ha resuelto la parte más complicada de la generación de código: las expresiones y el acceso a los datos.

En el cuadro 1 se puede observar que ya se están utilizando todas las instrucciones del código EML aquí mostrado salvo las últimas cuatro (*IR, IRF, LLA* y *RET*). Éstas serán las que se utilicen el mes que viene para generar código en las sentencias de control de flujo (si - sino - fin, mientras - fin, repetir - hasta, ...).

MODELO OSI MATERIALIZACIÓN SOBRE LAN's

María Jesús Recio

El modelo de referencia de Interconexión de Sistemas Abiertos, OSI (*Open Systems Interconection*) fue desarrollado por la ISO (*International Organization for Standardization*) a partir de 1978, intentando definir un conjunto de normas que permitieran interconectar diferentes equipos y posibilitar su comunicación. Este modelo define los servicios y los protocolos que posibilitan la comunicación, dividiéndolos en siete niveles diferentes, en el que cada nivel se encarga de problemas de distinta naturaleza interrelacionándose con los niveles contiguos, de forma que cada nivel se abstrae de los problemas que los niveles inferiores solucionan, para dar solución a un nuevo problema, del que se abstraerán, a su vez, los niveles superiores.

Los objetivos de este modelo son 4 básicamente:

- Interconectividad: definir las reglas que posibiliten la interconexión física y la transmisión de datos entre diferentes máquinas.
- Interoperabilidad: posibilita el trabajo interactivo entre máquinas, es decir, además de la transmisión de la información, la comprensión y el proceso de la misma.
- Independencia de instalación: el modelo puede ser implementado sobre cualquier arquitectura.
- Extremos abiertos: que la comunicación no se vea limitada a máquinas que trabajan con el mismo software.

Los niveles son: aplicación (nivel más alto), presentación. sesión, transporte, red, enlace y físico (nivel más bajo).

Cada nivel tiene definidas sus funciones, las funciones que el nivel inferior le ofrece, así como las funciones que debe facilitar al nivel inmediatamente superior. La forma de operar con esta implementación es similar a la jerarquía que se establece en unos grandes almacenes; un ejemplo partiendo desde el nivel superior puede ser:

Para que un disco esté colocado en la estantería correspondiente y así pueda ser comprado, es necesario que un reponedor lo halla cogido del almacén y lo halla colocado en la estantería. Para que el disco estuviera en el almacén un mozo de almacén ha tenido que desembalar las cajas que traen las mercancías y colocarlo en su sitio. Para que las cajas de mercancías hallan llegado hasta el almacén, es necesario que un camionero la halla traído desde la fábrica y las halla descargado. Como puede verse, cada figura tiene un papel a desempeñar; siempre supone que los elementos que necesita para desempeñar su función estén en su sitio, sin preguntarse cómo han llegado hasta él.

Como se puede apreciar, este método basado en la división por niveles, ofrece independencia a cada uno de los niveles y la posibilidad de evolución de cada uno de ellos según los avances en diferentes campos, de forma autónoma. En cualquier momento puede sustituirse uno de ellos, pues lo único que importa es que realice correctamente su función, independientemente de cómo lo haga.

Se puede decir, por tanto, que este modelo permite ver a una red como una sucesión lógica de capas, cada una de ellas envolviendo a los niveles inferiores y aislándolos de los superiores. Cada

A partir de este momento se va a ir profundizando en cada uno de los temas introducidos anteriormente. El modelo OSI es un modelo teórico de organización de las funciones necesarias para permitir la comunicación entre ordenadores, pero, ¿cómo se implementa esto sobre las redes locales

sistema, es decir, cada pila de niveles implementada en una máquina, es vista como si estuviese formada por una sucesión lógica de niveles (subsistemas).

Cada nivel, está compuesto por unos elementos, que son los que realmente realizan las funciones impuestas, y que se llaman *entidades*. Una entidad es capaz de enviar, procesar, y recibir información. A las entidades del mismo nivel, se les denomina entidades parejas (ver figura 1).

Para cada nivel se definen los siguiente conceptos: *servicio*, *función* y *protocolo*.

- *Servicio*: Un servicio no es más que el conjunto de prestaciones que un nivel y los niveles inferiores indirectamente ofrecen a un nivel superior. Estos servicios se proporcionan a través de las fronteras (límites) existentes entre dos niveles contiguos, llamadas *P.A.S.* (puntos de acceso a servicio). Estos *P.A.S.* representan los interfaces lógicos entre las entidades de dos niveles consecutivos de un mismo sistema. (ver figura 2).
- *Función*: Se trata de la forma de implementar los servicios que ofrece cada nivel, es decir, mientras que los servicios de un nivel se definen a nivel teórico, las funciones son la forma de materializar dichos servicios. (ver figura 3).

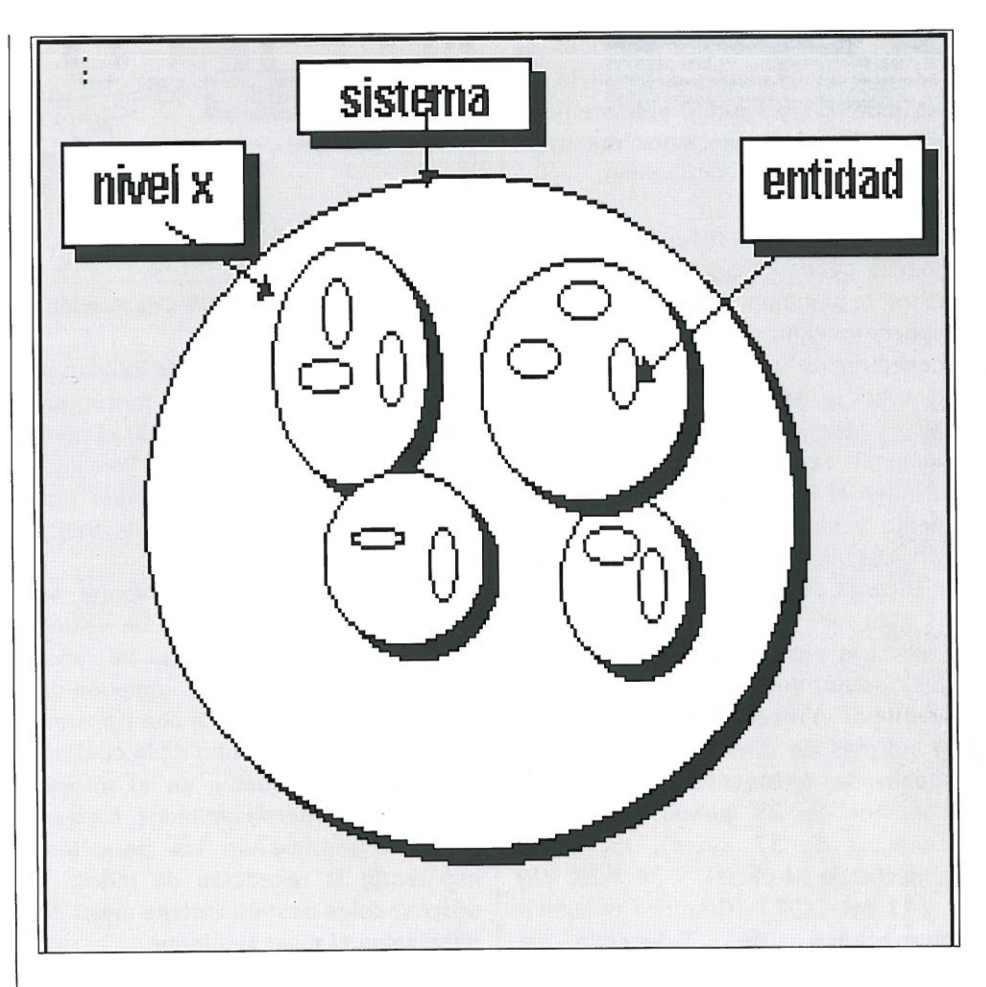


FIGURA 1: Esquema explicativo de Sistema, Nivel y Entidad.

El modelo OSI no es más que un modelo establecido, que se puede implementar de muchas formas: para cada nivel existen definidos muchos protocolos diferentes, por lo que se podrían crear sobre este modelo teórico distintas implementaciones prácticas.

Con esta estratificación se asegura la independencia de cada nivel, al definir los servicios que proporciona al nivel inmediatamente superior

- *Protocolos*: Los protocolos definen el conjunto de reglas que un nivel debe seguir para poder establecer comunicación con su nivel parejo en otra máquina. En el lenguaje coloquial, seria el conjunto de normas que definen una lengua. Para cada nivel se establecen protocolos diferentes. Téngase en cuenta, que los protocolos se imponen para poder establecer comunicaciones paralelas (entre iguales niveles de máquinas distintas) y no entre niveles consecutivos en una máquina.

Una vez conocidos todos estos detalles, se puede describir cómo se realiza la comunicación entre dos sistemas: El mensaje de la aplicación origen que quiere transmitirse desciende a través de todos los niveles de la torre, implementados cada uno con un protocolo específico, que le van añadiendo la respectiva información de control de la cabecera para el control de comunicaciones igualitarias horizantales. En el extremo receptor, la información de la cabecera se va eliminando en los niveles correspondientes, se procesa en la unidad funcional del nivel y los datos restantes suben al siguiente nivel donde tiene lugar una operación similar. La información asciende hasta alcanzar la aplicación de destino.

NIVELES OSI.

Después de hecho el repáso por las funciones de los niveles desde un punto de vista general, el lector debe

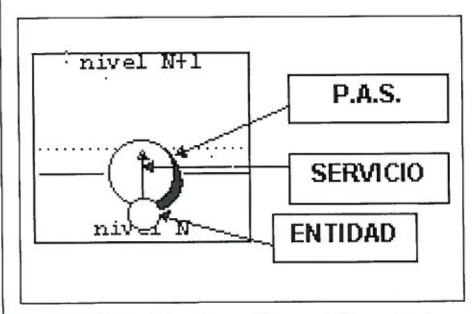


FIGURA 2: Explicación gráfica de los P.A.S.

entender cuáles son los objetivos de cada uno de los niveles del modelo por separado. En la figura 4 aparece reflejado un esquema de estos objetivos, que a continuación se detallan.

NIVEL FÍSICO: Es el nivel más bajo del modelo, define 4 aspectos: mecánico, eléctrico, funcional y procedural. En el aspecto mecánico se definen el tipo de conectores a utilizar. En el aspecto eléctrico la tensión asignada a los valores binarios 0 y 1. En el aspecto funcional se definen funciones que permiten el establecimiento, mantenimiento y liberación de la comunicación. Por último, el aspecto procedural se encarga de definir los procedimientos utilizados para el intercambio de datos. Las entidades parejas de este nivel están interconectadas, obviamente por el medio físico. Algunas de las normas de diferentes organismos establecidas a este nivel son: *ISO 2110* (conector de 25 pines), *ISO 4902* (conector de 37 pines), *ISO 2593* (conector de 34 pines), *V-24*, *V-28*, *V10* y *V-11* del CCITT (Comité Consultivo Internacional de Telegrafía y Teléfonos).

NIVEL DE ENLACE: Es el encargado de detectar y corregir los errores que se producen en la transmisión de los bits por el medio físico, para ello utiliza funciones como: Añadir a los datos un campo de control de errores o delimitar las tramas. Existen varios protocolos que se encuentran en este nivel, como pueden ser los protocolos orientados a bits (*HDLC*), y los protocolos orientados a caracter (*ASYNC*).

NIVEL DE RED: Tiene como funciones principales las siguientes: direccionamiento de los usuarios de la red, establecimiento/liberación de la conexión a nivel de red, encaminamiento y retransmisión de la información, multiplexación, segmentación y gestión de bloques, control de flujo, y transferencia de datos con prioridad.

NIVEL DE TRANSPORTE: Se encarga fundamentalmente de aceptar datos del nivel de sesión, dividirlos en unidades más pequeñas, si es necesario, y pasar estos al nivel de red, asegurando que todas estas unidades lleguen correctamente al otro extremo. Tambien es función de este nivel pro-

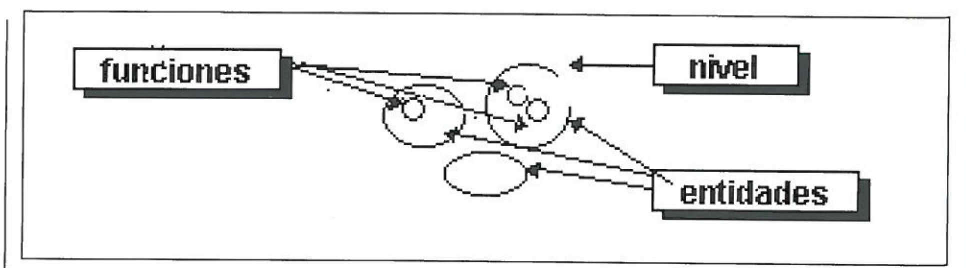


FIGURA 3: Explicación gráfica de Función.

porcionar un incremento de calidad al servicio de nivel de red, de forma que sea conforme al requerido por el nivel de sesión. Dependiendo del "desajuste" de calidades se determinará una clase distinta de protocolo de transporte.

Las conexiones de transporte se establecen entre entidades de sesión identificadas por direcciones de transporte. El tipo habitual de conexión de transporte corresponde a una transmisión sin error, por medio de la cual, se entregan los paquetes en el mismo orden en que fueron enviados. Esto se consigue numerando los paquetes, esperando la recepción de todos, y ordenándolos posteriormente antes de pasarselos al nivel siguiente.

Los protocolos de nivel de transporte son protocolos extremo a extremo, al igual que los de las capas superiores, es decir, una entidad de transporte en el sistema origen lleva una "conversación" con otra entidad parecida en el sistema destino. Los protocolos de los niveles inferiores son entre cada sistema y su vecino inmediato, y no entre los sistemas origen y destino, los cuales pueden estar separados por varios nodos de conmutación intermedios.

Un protocolo de extremo a extremo es aquel que hace transparente la realidad de no tener las dos máquinas que están conversando unidas directamente, de forma que sobre este protocolo ya se supone obviado dicho problema y se trabaja como si ciertamente las dos máquinas estuvieran unidas por un cable de forma directa.

NIVEL DE SESIÓN: Proporciona servicios de administración de la sesión y servicios de diálogo de sesión, para lo que gestiona el establecimiento de una conexión a su nivel, ofreciéndoselo a los niveles superiores.

NIVEL DE PRESENTACIÓN: Se trata de la capa del modelo que se encarga de transformar la información que le llega al formato que la capa de aplicación entiende. De esta forma, el nivel de aplicación no tiene que preocuparse de la representación de los datos que le llegan, por tanto, se puede decir que este nivel proporciona independencia respecto a la sintaxis en la que llega la información.

NIVEL DE APLICACIÓN: En realidad, este nivel consiste únicamente en una ventana para poder realizar el acceso al entorno OSI. Permite acceder a la información a las aplicaciones que la soliciten.

Si bien la transmisión efectiva de datos es vertical, cada una de las capas está programada como si fuera una transmisión horizontal

MÉTODOS DE ACCESO AL MEDIO.

El lector debe saber que el método de acceso al medio define las normas que la máquina debe seguir para utilizar el médio físico que soporta la red. Como ya se dijo, cuando varios ordenadores comparten un mismo medio para la transferencia de información, es necesaria la implantación de una política de acceso. Piénsese en un ejemplo real: a la hora de conducir,

como son muchos los coches que circulan por una misma carretera, es necesario establecer una serie de normas que faciliten la circulación; lo mismo pasa en el mundo de las comunicaciones.

Los principales métodos de acceso al medio son: *pooling*, *CSMA* y sus variaciones, y *token*. A contiuación, se ven las principales características de cada uno de ellos:

POOLLING: Consiste en ir preguntando a todos los nodos de la red si quieren transmitir; para implementar este método es necesaria la presencia de una estación que gestione esta política. Cuando una estación quiere transmitir, lo único que tiene que hacer es esperar a que le pregunten.

CSMA: (*carrier sense multiple access*). Esta política consiste en que, cuando una estación desea transmitir, lo que hace es escuchar el medio físico que implementa la red, (ver si circula señal portadora por él). Si el medio no está ocupado, entonces transmite, en otro caso espera un tiempo transcurrido el cual, lo vuelve a intentar. Se llama acceso múltiple, por que son muchos los nodos que pueden escuchar la red y comprobar que está libre en un mismo instante. Obviamente, este método presenta algunos problemas, las colisisones que se pueden producir. Para evitarlas, o por lo menos tratarlas, se han implementado dos versiones mejoradas del mismo: *CSMA/CD* (*.../collision detection*), en el que cuando una estación después de detectar el medio libre y empezar a transmitir, sigue escuchando; si detecta colisión, aborta la transmisión y espera un tiempo aleatorio para volver a intentarlo. La otra versión es *CSMA/CA* (*.../collisión avoidence*) evita las colisiones de la siguiente forma: cuando una estación quiere transmitir informa a todas las demás que va a hacerlo para que ellas no lo intenten. Este método no se utiliza ya que aumenta el tráfico en la red innecesariamente.

TOKEN: Consiste en introducir un testigo en la red, de forma que sea como el cajetin transporte de la información que circula por la misma. Cuando una estación quiere transmitir, lo que hace es esperar que el *token* vacío pase por ella. En ese instante introduce en el *token* la información a transmitir y, como en los taxis, pone a ocupado el *token* para que el resto de las estaciones así lo vean. El *token* circula por toda la red, cuando llega a su destino, éste hace copia de la información transmitida e introduce una señal de transmisión recibida en el *token*, y sólo cuando el *token* pasa otra vez por el origen es vaciado y puesto a libre para que el resto de las estaciones puedan utilizarlo.

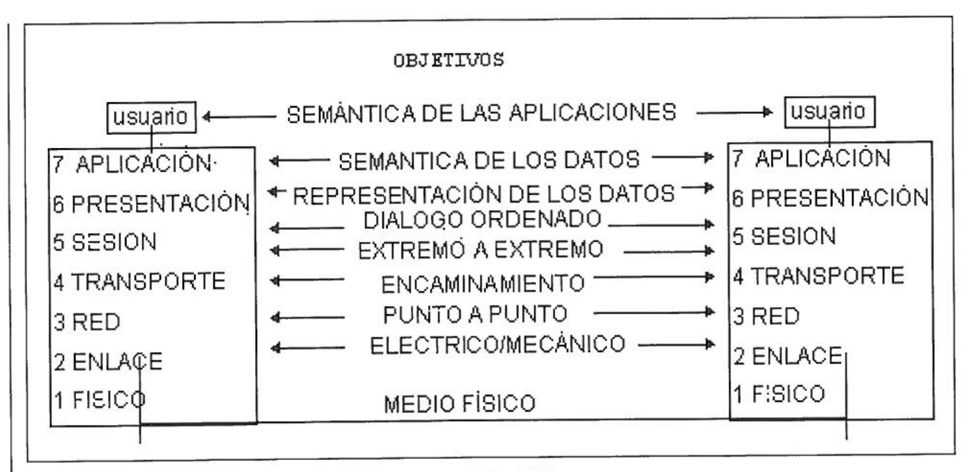


FIGURA 4: Objetivos de los niveles del modelo OSI.

TOPOLOGÍAS TEÓRICAS.

Como ya se dijo en el primer articulo, existen tres topologías puras teóricas definidas: bus, anillo y estrella. En la práctica se implementan topologías hibridas que mezclan varias de las topolgías puras para conseguir mejores prestaciones.

NORMAS 802.X.

Como ya se dijo al principio del artículo, este modelo es teórico y en base a él se han definido modelos prácticos con un protocolo concreto materializando todos y cada uno de los niveles, e incluso con alguna modificación de niveles, como es el caso del modelo empleado para redes de área local definido por el *IEEE* (*Institute of Electrical and Electronic Engineers*) en sus comites 802. Estos comités se han encargado de definir básicamente el nivel físico y el nivel de enlace, realizando algunas modificaciones sobre el modelo teórico OSI (ver figura 5). Para este modelo sólo se definen los 4 primeros niveles, por lo que suponen que las aplicaciones se apoyan directamente sobre el nivel de transporte.

La norma *802.1* es la más genérica de todas y define el nivel equivalente a los niveles de aplicación, transporte, y red del modelo OSI. La norma *802.2* define el subnivel *llc* (*logic link control*) del nivel de enlace. Las normas *802.3*, *802.4*, *802.5* y *802.6* definen el subnivel *MAC* (*control acces medium*) del nivel de enlace y el nivel físico. La figura 6 muestra gráficamente los elementos que contemplan dichas normas.

De la figura se deduce que existen definidos estándares sobre las arquitecturas de red más usuales; es posible encontrar otras arquitecturas diferentes, en la práctica, pero las únicas que están definidas son las que aquí aparecen.

Cada una de estas normas definen hasta los últimos detalles necesarios

La comunicación horizontal entre las entidades N+1, se realiza utilizando únicamente los servicios prestados por las entidades N

MODELO DEL IEEE PARA LAN's	MODELO OSI DE LA ISO
LLC	ENLACE
MAC	
FISICO	FISICO

FIGURA 5: Comparación del modelo IEEE y el modelo OSI.

para crear una LAN: tipo de cable, longitud entre segmentos, tipo de comunicación, velocidad de transmisión, etc. Otras característcas de estas normas, que no aparecen reflejadas en la figura 6 son:

802.3: Topología física en estrella o bus y lógica en bus. La distancia máxima entre estaciones en un solo segmento es de 500 m. La distancia entre estaciones debe ser valores múltiplos de 2,5 m. Los segmentos pueden conectarse mediante repetidores. Se permiten hasta 4 repetidores por segmento. Su rendimiento baja al aumentar el tráfico de la red. La comunicación se establece en banda base. El número máximo de estaciones en una red de este tipo es de 1024.

802.4: Topología física en bus y lógica en anillo. El testigo se pasa en un orden lógico en función de la dirección de cada estación. Cada estación conoce la dirección de la estación anterior y la posterior. No suele utilizarse en entornos de LAN. Su rendimiento no sufre alteraciones significativas al aumentar el tráfico.

802.5: Topología en anillo lógico y física en estrella. Máximo número de estaciones en un anillo 260, ampliables a través de un *bridge*, con un máximo de 8 *bridges*. Distancia máxima entre estaciones 700 metros. Su rendimiento no sufre alteraciones significativas al aumentar el tráfico. Transmisión en banda base.

LAS TOPOLOGÍAS EN LA PRÁCTICA.

Como topologías puras se definieron tres: bus, estrella y anillo, pero se ha visto que cuando se llevan a la práctica se hace diferencia entre la topología física que se implementa y la topología lógica.

La topología en estrella suele ser utilizada para implementar físicamente las otras dos topologías, aunque a nivel lógico se traten como en realidad son. Esto es debido a que la topología en estrella tiene una ventaja muy importante sobre las otras dos: cuando una estación se cae en esta topología, el resto de la red sigue operativa; esto no ocurre en los otros dos casos.

Para implementar una topología lógica en bus a través de una topología física en estrella se utiliza un *hub* al que se conectan todos las estaciones; aunque internamente el *hub* implemente un bus, externamente cada estación está conectada a él, por lo que si se cae el resto sigue funcionando.

Para implementar una topología lógica en anillo, a través de una topología física en estrella se utiliza una *MAU* a la que, de la misma forma, se conectan todas las estaciones. Este dispositivo forma internamente un anillo. Nótese, que si la estación que se cae es la que gestiona la red (en una red en anillo, existe al menos una estación que gestione el *token*).

Hay que destacar que la topolgía está íntimamente ligada al método de acceso al medio que se utiliza y, por tanto, también condiciona el dispositivo que se utiliza para la interconexión.

Las topologías tanto lógica como físicamente implementadas a través de una estrella tienen como dispositivo central la estación que gestiona la red, y que a su vez es el servidor de ficheros. Éstas suelen utilizar como método de acceso al medio la técnica de *Poolling*, analizada anteriormente. Este tipo de redes se utilizan cuando la comunicación se establece fundamentalmente contra el servidor y no entre las estaciones en sí. No existe ninguna norma sobre LAN's del *IEEE* que las defina.

También es conveniente citar que una LAN puede implementar varias topologías lógicas en todo su recorrido; mediante esta conjunción de topologías pueden conseguirse los denominados híbridos como por ejemplo una topología en arbol, en bus-arbol, etc.

PRÓXIMO NÚMERO.

En el tercer artículo de la serie se describirán los diferentes dispositivos que permiten interconectar y ampliar redes de iguales y diferentes topologías y arquitecturas. Además, se explicarán los motivos por los que es necesario segmentar una red, y la forma de hacer esto.

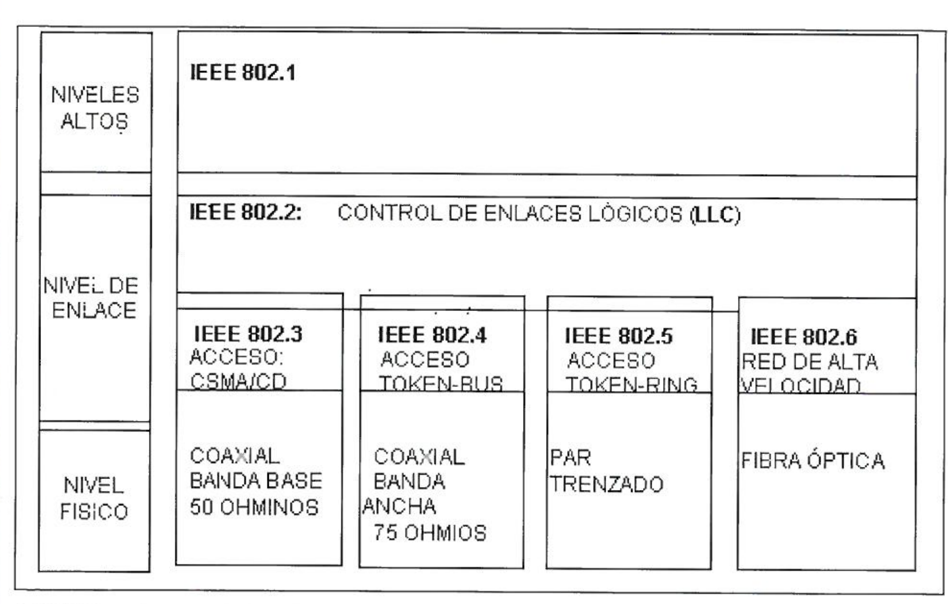


FIGURA 6: Normas de los distintos niveles.

VIRTUAL TENIS

TODO EL TENIS MUNDIAL EN UN CD-ROM DE IMPACTO...

...renos de juego creados en ... con nubes fractales.

Cuadro de partidos disputados en el torneo en curso.



...torneos se pueden jugar ... compañía de un amigo.



Control de los jugadores con teclado o Joystick.



...ibilidad de jugar el Grand Slam completo: permite grabar ...neos y temporadas a la mitad.

Virtual TENIS

POR SÓLO 2.995 PTAS. (IVA INCLUIDO)

4 JUGADORES · RED · MODEM · CABLE SERIE

8 413042 303190 00001

CONTIENE: PC

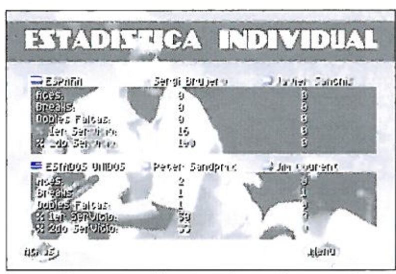
REQUERIMIENTOS : PC 386 o superior, 4MB memoria RAM, lector de CD ROM, tarjeta VGA, disco duro, ratón recomendado. Compatible con Sound Blaster y Gravis Ultrasound.

CON LA GARANTIA DE: DDM DIGITAL DREAMS MULTIMEDIA

ABETO EDITORIAL



...pueden disputar los cuatro torneos del Grand Slam.



Completas estadísticas de cada partido.



Control independiente de música y sonido.

- ...sibilidad de conectarse con distintos jugadores mediante red, ...ódem y cable serie.
- ...stema de menúes totalmente intuitivo.
- ...uda interactiva durante el desarrollo del juego.
- ...cil instalación en el disco duro.
- Jukebox para escuchar las distintas melodías que contiene.
- Cuatro diferentes superficies.
- Cada uno de los jugadores posee sus propias características.
- Permite elegir el número de sets que se desee disputar.
- Posibilidad de cambiar incluso el detalle gráfico.

CON LA GARANTÍA DE

DDM DIGITAL DREAMS MULTIMEDIA

...licita VIRTUAL TENIS enviando este cupón o llamando al teléfono (91) 661.42.11* de 9 a 14 y de 15:30 a 18:30, o por Fax: (91) 661.43.86

...eo que me envíen: ☐ **VIRTUAL TENIS por 2995 ptas. + 250 ptas. de gastos de envío.**

...ombre y apellidos..Domicilio..Población..........................

...ovincia..C.P ....F. de nacimiento........................Profesión..

FORMA DE PAGO:

Talón a ABETO EDITORIAL ☐ Contra-reembolso Teléfono .... Firma ,

Giro Postal nº..de fecha.........................

Tarjeta de crédito VISA nº ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐

...cha de caducidad de la tarjeta....................................Nombre del titular, si es distinto................................

Rellena este cupón y envíalo a: ABETO EDITORIAL C/ Aragoneses, 7. 28100 Pol. Ind. ALCOBENDAS (Madrid)

ABETO EDITORIAL

INTRODUCCIÓN AL MODO X

Santiago Romero, Miguel Cubas y Vicente Cubas

Actualmente, el modo 13h se ha convertido en la estrella de los modos gráficos de la VGA. Este modo basa su éxito en la simplicidad de su programación, además de sus 256 colores simultáneos en pantalla. Imagínese conseguir con su VGA resoluciones más altas que proporcionen a sus programas un aspecto profesional. Esto está ahora a su alcance gracias al MODO X

El modo 13h posee una resolución de 320 x 200 píxels, con 256 colores simultáneos, de 262.144 posibles cambiando los componentes DAC de la tarjeta (cambiando la paleta). Éstas son unas aceptables características en cuanto a colores y resolución horizontal, pero la resolución vertical de 200 píxels proporciona a los gráficos un aspecto alargado, más alto que ancho, de manera que un círculo en MCGA 13h parece mas bien una elipse achatada. La solución estaría en usar las tarjetas Super VGA, con resoluciones entre 640x480 y 1280x1024, a 256 y más colores, que proporcionan ratios (relación ancho:alto) de aspecto cuadrado. Posiblemente piense que estas tarjetas son definitivas, pero tienen grandes desventajas por ahora:

• Las tarjetas Super VGA, por lo general, son algo lentas, y lo que es peor, hay diferencias de velocidad entre ellas. Las altas resoluciones hacen que las operaciones de pantalla en SVGA se ralenticen usando el estándar Vesa. Además, programar cada tarjeta por separado resulta una incomodidad, debido al gran número de ellas (Trident,Tseng, etc).

• Si alguna vez ha parpadeado alguno de sus scrolles o sprites, imagínese mover bloques de 100x100 píxels que, en una pantalla de 1024x768, no son más que sprites "pequeños".

• Se acabó de dibujar en un segmento aparte y volcarlo a la memoria con rapidez. El sistema de planos de las SVGAs hace la programación más complicada.

• Las compresiones gráficas no consiguen evitar el abultado tamaño de las imágenes en el disco duro, a menos que se usen complejas compresiones (GIF, JPGE, etc).

A esas desventajas, se le une el impacto definitivo a favor del modo 13h: La programación del modo 13h hace que poner un punto sea una operación matemática para conseguir un offset o desplazamiento respecto al punto (0,0) de la pantalla. La programación de este modo gráfico debería ser familiar antes de iniciarse en Modo X, pero por si acaso se verán las bases.

Los gráficos que están en pantalla son guardados en memoria en forma de bytes, y el ordenador, cada vez que tiene que refrescar la pantalla, se dedica a leer de la memoria y redibujar los píxels antes de que el fósforo se extinga de cada uno de los puntos que componen el monitor. A partir del segmento A000h offset 0 de la memoria se encuentra la imagen binaria que representa la pantalla VGA. Debido a que el rango de valores de un byte está entre 0-255, se dispone de 256 colores, cuyos tonos pueden ser modificados cambiando el DAC, o lo que se conoce comúnmente como cambiar la paleta. Es muy sencillo calcular el offset de un píxel con la famosa fórmula:

*Offset = (y * 320) + x;*

o su variación ultrarápida:

Offset = (y << 8) + (y << 6) + x.

Con ella, se "saltan" 320 bytes por cada línea, y se le suma la coordenada X, para calcular el offset en memoria en el que colocar el byte al que corresponda el color del pixel, siempre en el rango 0-255.

Esto quiere decir que, si se coloca el número 15 (color blanco) en la posición A000:0000 de la memoria, aparecerá un píxel de ese color en la posición (0,0) de la pantalla. Para rellenar la pantalla de color negro (0) habría que rellenar 64.000 bytes a 0 en 0A000h. Hay que fijarse en un detalle que condicionó el descubrimiento del modo X. Si se usa un byte por píxel, 320x200 pixels

da un offset máximo de 64.000 (apróx. 64K). Pero resulta que TODAS las tarjetas VGA estándar tienen como mínimo 256 Kb de memoria o más. ¿Dónde están esos 192 Kb que faltan?

LA ORGANIZACIÓN PLANAR

Si se conoce algo sobre los planos, el lector sabrá que ésta es la razón del despilfarro de memoria. Todo viene por mantener la compatibilidad con las anteriores tarjetas. Los creadores de las tarjetas EGA (Enhaced Graphics Array) usaron un método denominado "planar", que toma su nombre del uso de planos. Como el lector sabrá, cada color está formado por tres componentes: Red, Green y Blue (R, G y B), que son los colores primarios cuya mezcla da el resto de colores existentes. El máximo color posible en una tabla de 16 colores es el 15 (rango 0-15). Ese número puede ser representado en binario justo por un nibble (4 bits, de tal manera que 15 = 1111b).

En vez de usar un byte por cada color, como en el modo 13h, se les ocurrió poner cada uno de los bits que indican el color (p. ej. 1011b) en un "plano" de la tarjeta, que podría compararse a 4 segmentos separados, para lo que usarían cuatro planos. El primer plano contiene los rojos, el segundo los verdes, el tercero los azules y el último la intensidad.

Como ejemplo, si en una tarjeta EGA se ponen, aleatoriamente, bits a 1 en el primer plano, aparecerían en la pantalla puntos rojos, que son los que contiene el primer plano. Para poner un punto blanco habría que poner un 1 en todos los planos en el bit correspondiente, ya que 15 = 1111 = blanco, siempre que no se haya cambiado la paleta. Con las combinaciones posibles (0000, 0001, 0010, ... ,1111) se dispone de los 16 colores que el DAC de la EGA dibujará en pantalla.

Así pues, ya se puede diferenciar entre modos de vídeo planares (como los de 16 colores y modo X) y lineales (como el 13h). Resulta además que, al crear la VGA, los fabricantes decidieron hacer un modo fácil de programar (13h), que resultó ser un modo lineal (todos los bytes de los pixels son consecutivos), pero con parte planar, ya que la VGA guardaba automáticamente esos bytes del segmento A000 en sus planos como en la figura 1, lo que acla-

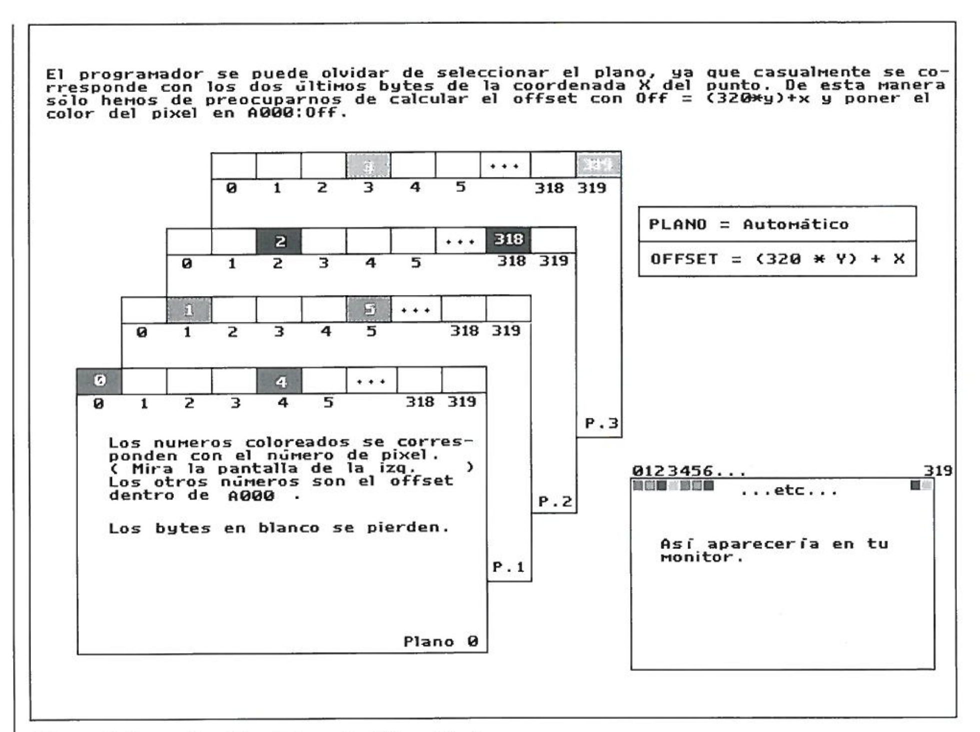


Figura 1. Organización del modo 13h estándar.

raba lo de los 192K que faltaban. No han desaparecido, sino que el sistema de planos nulo no los utiliza. El programador no tiene que preocuparse de los planos en el modo 13h, ya que son gestionados por la propia tarjeta, ahorrando al programador su cálculo, pero haciendo pagar un alto precio por ello: se pierden 196.608 bytes (192K) que, en modo X, supondrán 3 páginas más de vídeo para dibujar, lo que favorece cosas como la animación por *page flipping*.

PASANDO AL MODO X

A cierto programador, llamado Michael Abrash, lo de los 192K desaparecidos en la VGA no le dejaba dormir por las noches. Así que comenzó a experimentar con la VGA. En particular, hubo un registro que le llamó mucho la atención, y se puso a trabajar sobre él.

Michael Abrash descubrió que había un bit en la VGA que, cuando estaba a cero, forzaba a la tarjeta a usar planos de bits, como en 640x480 y la mayoría de resoluciones EGA, y que se ponía a 1 cuando los modos eran lineales (como el 13h). Ese bit es el llamado CHAIN-4, porque encadena los cuatro planos. De manera que ya se tiene un paso adelante hacia el modo X: Inicializar el modo 13h y cambiar el bit en cuestión, cosa que se hace enviando un valor a un determinado puerto (ver ejemplos de inicialización). Con esto se fuerza a la VGA a gestionar el modo 13h como un modo de 4 planos, "desencadenándolos" o liberándolos del modo lineal. De ahí el termino unchained. Pero eso no basta para entrar en el modo X. Hay que ajustar otros registros en la VGA como sincronización, refresco, etc., que evitan parpadeos y que son necesarios para la correcta visualización de las imágenes, así como para que la propia tarjeta reconozca el nuevo modo de gestión de la memoria.

"Pero si no hay modo linear, sino planar, la programación del modo X con respecto a poner píxels, etc. se hace más complicada", pensará el lector. Acierta, pero el lector descubrirá que también hay fórmulas de offset parecidas a las que se usan en modo 13h, que automatizarán la tarea de colocación de pixels. Lo más importante, por ahora, es comprender el modo de organización unchained para crear las primeras rutinas gráficas.

Así que, para comenzar, habrá que mirar el ejemplo 1, que sirve para ilustrar al inicialización de 320x200x256 unchained que, aunque todavía no es el llamado modo X, tiene sus mismas características y se suele usar tanto como el 320x240. En el disco que acompaña a la revista, hay un programa llamado

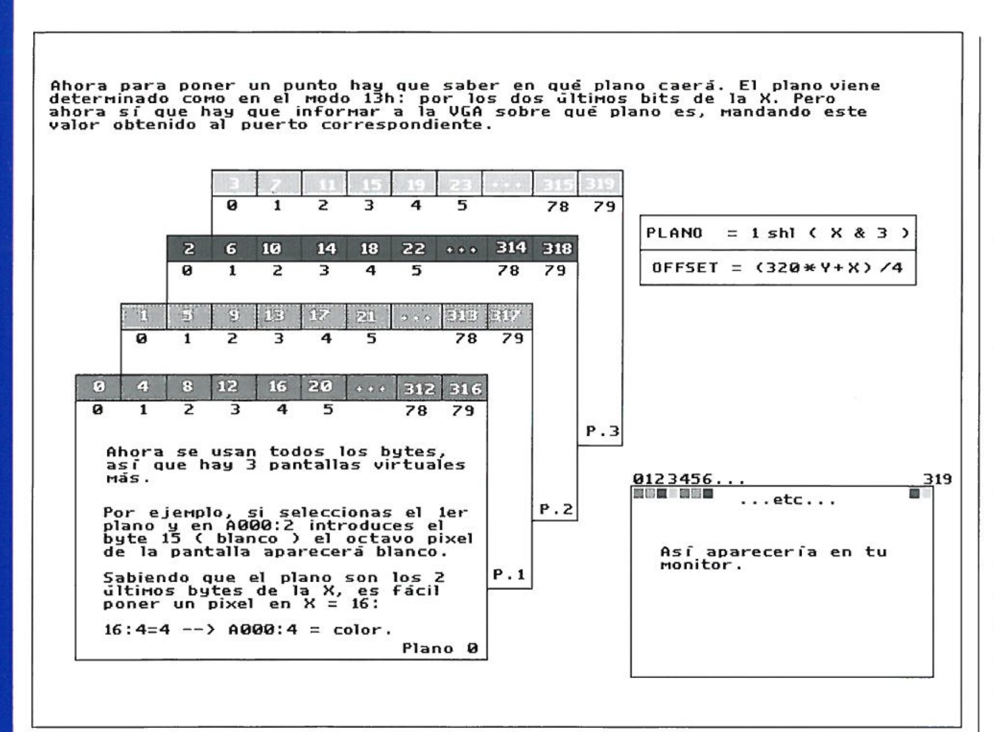


Figura 2. Organización del modo X planar o unchained.

RATIO.EXE que muestra la diferencia de resolución entre el modo 13h y el modo X. Las rutinas gráficas que se crearán funcionan tanto con 320x200X como con 320x240X. Si se posee una SVGA, dado que son compatibles con la VGA, también se podrá disfrutar del modo X. Hay que hacer notar que hay algunos monitores que no soportan algunas resoluciones X, como 360x400 o 360x360, sobre todo los antiguos.

Lo que hace el programa es inicializar primero el modo 13h, para después reprogramar algunos registros de la VGA, de manera que, después de apagar el bit CHAIN 4, la tarjeta inicie el nuevo modo gráfico 320x200X. La inicialización del modo 320x240x256, o MODO X, se reduce a una modificación del total vertical register, además de registros de polaridad y retrazo, para que redibujen las nuevas 40 líneas extra de pantalla y para que las reconozca la tarjeta. Pero eso se verá en la próxima entrega, dedicada principalmente a la programación de los registros más útiles para modos unchained de la VGA.

Como lo que interesa de momento es la teoría, se ha usado ensamblador, ya que se asume que, quién se quiere meter en modo X es porque ya conoce a fondo el modo 13h y algo de Ensamblador.

Además de 320x200, es posible conseguir resoluciones tales como 320x240, 360x400, 400x600, 256x256 y otras similares. Pero lo que más interesa ahora es comprender la lógica de planos de los modos X. Una vez sea entendido esto, desaparece ese halo de misterio que rodea al modo X y como comienza a ocupar un lugar importante en sus futuras creaciones. En el ejemplo 2 se puede ver la rutina que inicializa el MODO X con 320x240 píxels de resolución.

LA ORGANIZACIÓN DE LOS MODOS UNCHAINED

Ahora llega la parte más importante. Si se comprende la organización del modo 13X y la manera de poner un píxel en pantalla, es posible crear una primera librería del modo 13X (320x200X256_X y 320x240x256_X). Después de esto, dibujar sprites, líneas y polígonos será sencillo, y se hará de manera muy parecida al modo 13h. Primero puedes mirar la organización del modo 13X (y similares) en la figura 2.

El sistema de planos es algo más complicado que el lineal del modo 13h. Como se puede ver en la figura, en el primer plano de la VGA se guardan los pixels 0, 4, 8, etc.; en el segundo 1, 5, 9, etc. y así con los 4 planos. Entonces, para poner un píxel en pantalla, basta con seleccionar el plano correspondiente y poner en A000:offset el valor del color del píxel. Los planos están guardados en la memoria que se acaban de aprovechar (los 192K) y puede usarse el segmento de la VGA (0A000h) para comunicarle a la tarjeta los cambios que se desean hacer en estos planos. Esto se reduce a colocar en el segmento A000h bytes que indiquen los colores de los pixels. Tomando en cuenta esto datos, en PutPixelX se debe:

A. Seleccionar el plano correcto.
B. Calcular el offset desde A000h.
C. Usar Stosb para poner el punto.

SELECCIONAR EL PLANO CORRECTO

Para seleccionar el plano en el que escribir en una VGA, sólo hay que mandarle el índice o función a realizar (en este caso cambiar de plano) al puerto 0x3c4 y el número del plano al puerto 0x3c5. Esto se puede hacer de manera muy sencilla:

```
outportb( 0x3c4 , 0x02 )
 /* función de seleccionar plano */
outportb( 0x3c5 , 1 )
 /* plano 1 */
```

o de manera "todo-en-uno", aprovechando que outport(port , valor); de DOS.H manda a port el byte alto del entero y a port + 1 el byte bajo, al igual que hace out de ensamblador.

```
outport( 0x3c4 , 0x0201 );
 /* los 2 bytes en uno */
```

Pero, ¿cómo se sabe en qué plano cae el píxel? Fácil: si el lector se fija en la figura 2, en cada plano los pixels aumentan de 4 en 4. Es tan sencillo como que los dos últimos bytes de la coordenada X lo indiquen, porque están entre 0 y 3. Ese número se podría saber en C usando plano = 1 << (X % 4) y en Pascal con plano = 1 SHL (X MOD 4), pero es mejor usar algo que se pueda transportar a ensamblador y que sea más optimizado que el operador %, pues éste implica una división y el posterior cálculo del resto:

```
Plano = 1 << ( X & 3 );
 /* EN C/C++, << = SHL y & = AND */
```

Y con esta fórmula que depende de la X ya se calcula el plano a activar. También se pueden activar los 4 planos al mismo tiempo, por lo que, al colocar un valor en memoria, aparecerían cuatro pixels consecutivos, algo bueno para dibujar líneas horizontales y rellenar polígonos. Los programadores más avanzados se habrán percatado de que el resultado de 1 << (X & 3) es un

número superior a 1, ya que hay que activar algún plano.

Los últimos 4 bits del byte que se le envían a la VGA le indican qué planos activar, según los bits estén activados o no. Así, si se le envía el byte 00000100b activará el tercer plano, ya que está activado el tercer bit.

Lo que se hace en la fórmula es colocar un 1 en el último bit (00000001b) y desplazarlo hacia la izquierda, hasta que el 1 se coloque sobre el plano que realmente se desea activar. Por ejemplo: supóngase que se tiene un pixel que ha de dibujarse en la coordenada (2,0). El resultado de x&3 será 2 (el plano en el que está). Ahora se coloca el último bit a 1, 0001b, y se desplaza a la derecha el número obtenido, cuyo resultado sería 0100b. Éste es el valor que se debe enviar al puerto 0x3c5 y que le indica a la VGA el plano que se desea activar. No se puede usar sólo x&3 porque, en el caso de que el resultado fuera 0 no se activaría ningún pixel (sería mov al, 0) mientras que lo que interesa es activar el pixel 0.

Debe recordarse que SHL a, b desplaza los bits del operando a, y lo hace b veces hacia la izquierda:

SHL 00111010b, 1 = 01110100b

La forma de activar los 4 planos es enviar como número de plano el valor 15 (1111b) porque, al tener activados los 4 bits, la VGA asume que se va a escribir en los 4 planos, de manera que al colocar, por ejemplo, el byte 1 en A000:0000 aparecerán cuatro pixels consecutivos con el color azul. Hay que hacer notar que, hasta que no se seleccione otro plano, la VGA mantendrá siempre activado el último plano seleccionado.

OFFSET EN MODO 13h	OFFSET EN MODO X
Off = (320 * y) +x	Off = (y*80) + (x/4)

Figura 3. Cálculo del offset en función de las coordenadas del punto.

EJEMPLO 1: INICIALIZACIÓN DEL MODO 320X200 UNCHAINED

```
/*  Ejemplo de inicialización del modo 13X */

#include <dos.h>                          /* outport */

#define SEQU_ADDR 0x3c4
#define CRTC_ADDR 0x3d4

void Set_320x200X ( void );

void Set_320x200X ( )
 {
   asm  mov ax , 13h                      /* inicializamos modo 13h estándar */
   asm  int 10h

        outport( SEQU_ADDR , 0x0604 );    /* ponemos el bit CHAIN-4 a 0 */
        outport( CRTC_ADDR , 0xE317);     /* ajuste del modo de palabras */
        outport( CRTC_ADDR , 0x0014);     /* ajuste del modo de dobles palabras */
        outport( SEQU_ADDR , 0x0F02);     /* seleccionar los cuatro planos para...*/

   asm {                                  /* ...borrar la pantalla */
         mov ax , 0xA000
         mov es , ax
         xor di , di
         xor ax , ax
         mov cx , 32000
         rep stosw
        }
 }
```

CALCULAR EL OFFSET DESDE A000

En MCGA estándar había que saltar 320 bytes por cada línea, más la coordenada X del punto, para calcular el desplazamiento desde A000. En modo X, el offset se calcula de forma similar:

Se sabe que la resolución horizontal de la pantalla es 320. Si hay cuatro planos, cada uno de estos planos guardará 320 / 4 = 80 pixels por línea. Es decir, al dividir por 4, ya se calcula el offset de la X, y por fin, si se saltan 80 bytes por cada línea, el resultado es el punto deseado.

Sobre la multiplicación por 80, hay que hacer notar que, en modo X, se dispone de 4 páginas, y pueden colocarse como se quiera, ya sea las 4 en sentido vertical, horizontal y mixto; teniendo 2 páginas de ancho por 2 de alto. El criterio de selección de un sistema u otro depende del tipo de programa o efecto a crear. Es decir, si se desea realizar un scroll de 4 pantallas horizontales, lógicamente se tendrá que inicializar el modo 4x1. En este caso, habría que saltarse 320 píxels por cada línea, ya que las 4 pantallas está colocadas una a continuación de la otra. Este tema será el desarrollado principalmente en el próximo número, donde se explicará con más detenimiento. Volviendo con el modo X colocado en 1x4 (1 pantalla horizontal, 4 hacia abajo), se tiene que, después de haber seleccionado el plano correcto, el cálculo se reduce a la fórmula usada en la figura 3.

Como ejemplo, imagínese que se coloca en A000:0000 el color 15 (Blanco). Si antes de poner el valor en memoria se selecciona el plano 1, aparecerá en la posición (0,0) un pixel blanco. Si el plano seleccionado era el 2, aparecerá en (1,0) y si se seleccionasen los 4 mediante el valor 1111b, aparecerían 4 pixels blancos (0,0), (1,0), (2,0) y (3,0). Fíjese que cada offset de la memoria sirve para poner 4 puntos, indicándole el plano correcto. Por eso, en vez de 320 píxels se necesitan 320/4 = 80. De esta manera,

80*200 pixels da 16.000 bytes. Como el segmento de la VGA es de 65.536 bytes, se dispone de 65.536 / 16.000 = 4'1 páginas gráficas donde dibujar y entre las que hacer scroll. Fíjese también que, en el caso del modo 320x240, en vez de tener 4,1 páginas, se dispone de 4 páginas justas, debido a las 40 líneas extras.

PONER EL PUNTO EN PANTALLA

A continuación se va a Desarrollar la primera rutina PutPixelX para hacer las primeras pruebas. Ésta se encargará de recibir las coordenadas X,Y y el color , calculará el plano y el offset del pixel y, finalmente, lo colocará en pantalla, siguiendo los pasos que se acaban de ver.

```
void PutPixelX( int, int, char );
void PutPixelX( int x, int y, char color )
{
unsigned int offs;
/* función seleccionar plano */
outportb( 0x3c4 , 0x02 );
/* esto calcula automáticamente el
plano
a partir de los 2 últimos bits de la
coord.x */
outportb( 0x3c5 , 0x01 << ( x & 3 ) );
/* calcula el offset */
offs = ( 80 * y  ) + (x / 4) ;
asm {
- mov ax , 0xa000
- mmov es , ax
- mov di , offs
- mov al , color
- stosb
/* pone el punto */
    }
}
```

Multiplicar por 80 y dividir por 4 puede ser profundamente optimizado mediante instrucciones de desplazamiento SHL y SHR (<< y >>) para acelerar la colocación de los puntos. La división entre 4 se puede sustituir por x>>2 y la multiplicación por 80 es los mismo que multiplicar por 16 más la multiplicación por 64. Aunque parezca algo complicado, puede sustituirse la línea

*offs = (80 * y) + (x / 4);*

por su variación optimizada:

offs = (y << 6) + (y << 4) + (x >> 2);

PRÓXIMA ENTREGA

En el próximo artículo se optimizarán las rutinas y se profundizará en el modo X de manera que cada lector explote el modo unchained que más le interese a sus juegos o programas (320x200, 320x240, 256x256 ...). Además, se proporcionará un listado de los puertos de los registros y ejemplos

EJEMPLO 2: INICIALIZACIÓN DEL MODO X, 320X240 UNCHAINED

```
/*   Ejemplo de inicialización del modo X */

#include <dos.h>                          /* outport */

#define SEQU_ADDR 0x3c4
#define CRTC_ADDR 0x3d4

void Set_320x240X ( void );

void Set_320x240_X( )
     {
     Set_320x200X( );      /* primero inicializamos 13X */

     outportb(0x3C2, 0xE3);
     outport(0x3D4, 0x2C11);        /* permitir escritura */
     outport(0x3D4, 0x0D06);        /* Total vertical */
     outport(0x3D4, 0x3E07);        /* Overflow Register */
     outport(0x3D4, 0xEA10);        /* Vertical retrace start */
     outport(0x3D4, 0xAC11);        /* Vertical retrace end y Write prot. */
     outport(0x3D4, 0xDF12);        /* Vertical display enable end */
     outport(0x3D4, 0xE715);        /* Start vertical blanking */
     outport(0x3D4, 0x0616);        /* End vertical blanking */

     }
```

BIBILOGRAFÍA

- Tweak v1.x de Robert SchMidt. Permite crear modos unchained.
- XIntro.txt del mismo autor.
-"Power Graphics Programming" de Michael Abrash, de QUE / Programmer's journal.
- XSharp v 5.1 . Rutinas para 3-D en modo X.
- PC- Games Programmer's Encyclopedy y SWAG 3.0.

de usos de scrolles hardware, subiendo el listón hasta un nivel más profesional. El objetivo es adquirir suficientes conocimientos como para desarrollar programas en otros modos (256x256 o modo q, 360x400, etc.) experimentando con los registros.

Como proyecto, se propone al lector que realice una función GetPixel (x , y); que devuelva el color de un píxel, partiendo de que, para poder leer de un plano, hay que usar, en vez de los puertos anteriores, los siguientes: (y que el offset se calcula de la misma manera).

```
/* Función leer de plano */
outport( 0x3CE , 0x04 );
/* plano a leer ( 0 - 3 )*/
outport( 0x3CE+1, plano );
```

El plano se calcula con plano = x & 3 y no con 1 << (x & 3) porque el plano de lectura ya no es superior a 1, y el valor 0 representa el primer plano. Esto es debido a que no se pueden leer varios planos al mismo tiempo, y por eso se optó por usar el método más sencillo de x&3, sin desplazamientos.

La función ya está implementada en la librería, pero es aconsejable que se intente su creación si se quiere desarrollar librerías para otros modos unchained, para ir adquiriendo práctica, ya que esta serie de artículos cubrirá sólo los principales modos. La librería está creada como un fichero *.H (MODOX.H) y para su utilización, basta con que el fichero esté en el mismo directorio que el fichero que se está compilando, y que se coloque al principio de los programas la línea #include "modox.h".

Para consultas sobre el tema, puede contactar con cualquier miembro de Compiler Software en las direcciones indicadas en los ejemplos del disco o en la propia revista.

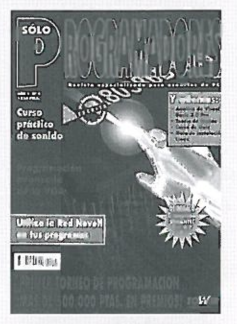
CÓMO SUSCRIBIRSE A



SÓLO PROGRAMADORES
Revista práctica para usuarios de PC









































scríbase enviando este cupón por correo o fax (91) 661.43.86, o llamando al teléfono (91) 661.42.11 Horario 9 a 14 y 15:00 a 18:00 h.

)eseo suscribirme a la revista SÓLO PROGRAMADORES acogiéndome a la siguiente modalidad:

☐ Suscripción: 1 año (12 números) **por sólo** 11.950 ptas. (ahorro 20%). ☐ Estudiantes carreras técnicas: 8.950 ptas. (ahorro 40%)

ESTA OFERTA ANULA LAS ANTERIORES, DESCUENTOS NO ACUMULABLES.

Iombre y apellidos.......................................Domicilio.......................................

'oblación.......................................C.P ....Provincia......................Telf...........................Profesión...................

ORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº ____ ____ ____ ____

Fecha de caducidad de la tarjeta...............Nombre del titular, si es distinto.......................................

☐ Domiciliación bancaria.

Señor Director del banco ....

Población ....

Ruego a vd. que se sirva cargar en mi ☐ cuenta corriente ☐ libreta

: ahorro número.......................................

recibo que le será presentado por TOWER COMMUNICATIONS, S.R.L.

mo pago de mi suscripción a la revista SÓLO PROGRAMADORES.

CODIGO CUENTA CLIENTE

ENTIDAD	OFICINA	DC	Nº CUENTA

Firma:

☐ Contra-reembolso del importe más gastos de envío.

☐ Cheque a nombre de TOWER COMMUNICATIONS S.R.L., que adjunto.

☐ Giro Postal (adjunto fotocopia del resguardo).

Rellena este cupón y envíalo a:
TOWER COMMUNICATIONS SRL
C/ Aragoneses, 7
28100 Pol. Ind. ALCOBENDAS (Madrid)

DISTRIBUCIÓN DE MEMORIA

David Aparicio

Este mes se va a revisar el núcleo más de cerca. Se verá el modelo de memoria que emplea el popular sistema operativo, desde que se arranca hasta que queda ejecutando de forma normal. Además, se verá una útil lista de los parámetros de configuración que Linux entiende en su inicio

Como ya se ha venido realizando en los últimos números, se va a analizar nuevos detalles del mismo núcleo. Para algunos lectores, interesados en el aspecto útil y funcional más que en la cruda programación de Unix, este artículo puede despertar poco interés. Sin embargo, y quizá dando la razón a la expresión "el saber no ocupa lugar", es posible afirmar que incluso a partir de la revisión de este tema puede encontrarse información muy útil para resolver posibles problemas de arranque de Linux.

Por las propias cartas recibidas de los lectores, a los que agradecemos desde aquí el apoyo y seguimiento demostrado, se deduce que parte de los problemas que impiden comenzar a disfrutar del uso de Linux parten del fallo en la detección del hardware en el arranque.

Para evitarlo, el usuario debe indicar parámetros que informen del hardware "especial", tanto si usa LOADLIN, LILO o la línea de '*boot*' en el mismo instante del arranque.

ARQUITECTURA DEL 386

Antes de comenzar, quizá sea conveniente explicar lo que ofrece un micro 386 o superior para que un sistema operativo moderno ejecute con comodidad.

En primer lugar, un 386 tiene cuarto niveles de ejecución que, por símil con una estructura de capas en "hojas de cebolla", seguirán la nomenclatura "externo" a "interno" para indicar de menor a mayor nivel de privilegio. Cada uno de estos niveles sólo puede ser interrumpido por otro más prioritario. Se explicará para qué sirve esto de forma inmediata.

En segundo lugar, existen tres modos de ejecución en el micro:

• El modo "*real*" es en el que nace tras un reset, equivale a un 8086 que simplemente tiene registros de 32 bits y es muy rápido. Es la forma en la que DOS utiliza el procesador, y casi la única justificación para que este comportamiento todavía esté implementado en el micro.
• El modo "*protegido*" permite la capacidad multitarea del procesador, con los cuatro niveles de ejecución mencionados antes, ejecución independiente de tareas, acceso controlado a dispositivos, y otros. Es el modo usado por verdaderos sistemas operativos, como OS/2 Warp, Windows NT, y Linux.
• El modo "*virtual 86*" permite, una vez que el modo protegido está activado, sesiones que corren en modo 8086 compatible, y que está pensada para compatibilizar sesiones DOS dentro de un sistema multitarea. DOSEMU usa este modo.

Sorprendentemente, incluso DOS estuvo a punto de ser un sistema multitarea, utilizando el modo protegido del 386. En la versión 4.0 ya hubo un intento de bendecir a este sistema, y existen muchas funciones indocumentadas que así lo indican. Por razones comerciales (la llegada de Windows) esta tendencia se abortó, y hoy día el sistema sigue siendo el de siempre. Sólo el programa EMM386 pasa la máquina de modo real a modo virtual 86 (protegido), con una sesión DOS ejecutando a nivel 3 (externo) y una parte del driver en nivel 0 (interno), la cual se emplea únicamente para mos-

trar un mensaje cuando se produce un fallo fatal en DOS.

En modo protegido, se utilizan dos características vitales para la multitarea: el direccionamiento lógico y la memoria virtual. A grandes rasgos, el primero consiste en disponer de un descriptor (*segmento*) por cada proceso en ejecución, que asigna direcciones lógicas (las que ve el programa, contiguas y comenzando en la dirección cero) a direcciones físicas (la RAM existente, no necesariamente contigua). La memoria virtual va a permitir a un programa intercambiar partes de sí mismo (*páginas*) entre disco y memoria, maximizando de este modo la forma de utilizar la memoria.

El código que intercambia las páginas entre disco y RAM está en el núcleo, y ejecuta a nivel más privilegiado que el de los procesos paginables. Cuando uno de éstos intenta ejecutar código que no se encuentra en memoria, se produce una excepción y el nivel más interno toma control, haciendo el intercambio necesario.

El mapeo de memoria lógica y física permite que un proceso esté localizado en cualquier parte de la RAM (incluso en varios trozos de la misma) desde el punto de vista del núcleo, mientras que desde su propio punto de vista se encuentra en unas direcciones fijas, lineales y conocidas ya en la compilación del mismo.

ARQUITECTURA DE LINUX

Con todo lo visto anteriormente ya se puede concretar lo que hace Linux. El núcleo ejecuta al mismo nivel de privilegio todo él, con lo que no admite excepciones mientras ejecuta, no puede ser paginado, y por ello debe estar completamente cargado en memoria.

Para paginar el núcleo se necesitaría un diseño por capas "en cebolla", donde la más interna se denomina *microkernel*. Versiones muy modernas de Unix (como *Mach)* siguen esta política, y se basan en el máximo aprovechamiento de la memoria. Sólo el tratamiento de interrupciones, excepciones, la paginación y el planificador se encuentran en el nivel interno. El resto, drivers incluidos, se cargan bajo demanda en capas de programa menos privilegiadas. Existe una versión de Linux portada para esta arquitectura (Mach Linux server), que ha dado lugar a una alianza entre Apple y OSF, como se explica en el cuadro 1.

Tras la inicialización, se llama a la primitiva "fork" por primera vez, se crea una copia del núcleo, que pasa a ejecutar como un proceso independiente

El núcleo de Linux ejecuta al mismo nivel de privilegio todo él

Por último, la forma en la que se pasa de un nivel de privilegio a otro es mediante:

- *excepciones*: Por causas involuntarias al proceso, el núcleo debe atender un evento para que el código menos privilegiado pueda proseguir.
- *puertas*: Por voluntad del proceso, se invoca un servicio del núcleo mediante un punto de entrada autorizado. Se ejecuta en un nivel más privilegiado, hasta que se retorne del servicio.

Volviendo al Linux clásico,hay que repetir que es monolítico, se carga físicamente por encima del primer mega, y tiene un mapeo identidad entre memoria física y lógica. Esto último quiere decir que ve las direcciones de memoria tal y como se encuentran realmente en la máquina. Esto es muy útil, puesto que el hardware emplea ciertas zonas para poder ser accedido: La memoria de vídeo, memoria de tramas ethernet, y algunos otros casos ocupan direcciones en el rango de 0x0A0000 a 0x0FFFFF (entre 640Kb y 1 Mb). Las extensiones de 32 bits de la BIOS (funciones APM y PCMCIA) se mapean en el límite superior de 4 Gb. Así, el mapeo identidad de esta memoria facilita la programación de drivers, ya que se ve a esta memoria justo donde debería estar.

Por contra, los procesos tienen una visión diferente. Se les asigna un espacio de 4 Gb de memoria virtual para ejecutar, dividida en el primer Gb para código, el segundo y tercero para datos, pila y memoria dinámica, y el cuarto para código del núcleo. Este último segmento sólo es accesible mediante puntos de acceso, o *puertas*, formadas por las llamadas al sistema. Como ya se ha explicado, no es paginado y es compartido por todos los procesos. El código propio del proceso es paginable, compartible (ya que varias instancias de un mismo programa ocupan la misma memoria física), y no escribible. Por último, los datos y pila son únicos a cada proceso, escribibles por él y paginables.

Como puede observarse, la capacidad para que todos los procesos vean cosas diferentes en las mismas direcciones, compartiendo incluso trozos de código, se debe a la asignación de direcciones lógicas mediante descriptores de tareas del 386. Por limitaciones del microprocesador, Linux sólo puede tener 126 tareas simultáneas, más que suficiente para un ordenador personal.

Cuando un proceso se carga, se le dan inicialmente tres páginas en memoria, una para código, otra para datos, y otra para pila. Es decir, que con 3 Kb, un programa puede empezar a ejecutar. Conforme se va accediendo a otras zonas del mismo, se cargan nuevas páginas desde disco (política denominada "*page on-demand*"). Si todos los procesos en ejecución consi-

guen ocupar toda la RAM, se vuelven a dejar en disco (swap) las páginas no usadas, mediante algoritmos específicos. Esto ralentiza la ejecución, pero es adecuada si se trata de un hecho puntual. Si, por el contrario, el disco está continuamente en funcionamiento, quizá ha llegado el momento de una ampliación de memoria.

ARRANQUE DEL SISTEMA

Todo este montaje de páginas, asignación de direcciones, descriptores y demás, deben ser construidos en algún momento. Cuando el PC carga el sistema operativo tras el encendido, le da control en modo "real". En este momento, Linux debe inicializarse, construir las estructuras de datos y descriptores adecuados, antes de seguir ejecutando en modo protegido.

En */usr/src/linux*, disponible si se han instalado los fuentes del núcleo de la serie K, es posible encontrar todas las referencias que se va a realizar a continuación. Se analizará la versión de núcleo 1.2.13, por ser la oficial y más estable en este momento. La figura 1 representa un esquema de todo lo que vamos a indicar a partir de aquí.

En *arch/i386/boot* se encuentra el código de inicialización del núcleo. El fichero *bootsect.S* contiene el cargador, ocupando un máximo de 512 bytes, que es lo único que la BIOS trae a memoria cuando cede el control.

Linux permanece ligado a la exsitencia de una BIOS para poder arrancar, puesto que forma parte del estándar del PC

Comienza a ejecutar en la dirección 0x7C00, y lo primero que hace es copiarse a sí mismo en la dirección 0x90000, ejecutando desde allí. Esto es justo por encima de los 512 Kb de RAM, lo cual indica que se necesita un sistema de 640 Kb para empezar con Linux. Sólo los poseedores de 386SX necesitan entrar en estas disquisiciones, puesto que los sistemas actuales están todos equipados correctamente.

Entonces se utilizan llamadas a BIOS, en modo real 16 bits, para cargar otros trozos de núcleo en 0x90200, (justo tras él), y el resto en 0x10000, tras los primeros 64 Kb de RAM. Como se esta utilizando la BIOS en este procedimiento, y dado las tablas y datos que ésta emplea se almacenan en las primeras posiciones de la RAM, si se aprovechára esta zona para cargar el núcleo, probablemente "se colgaría" la máquina en el arranque. Por otra parte, como se dispone de una zona entre 0x10000 y 0x90000 (512Kb), el tamaño de *vmlinuz* (o la imagen de arranque) no puede superar este tamaño para poder ser cargado.

Tras finalizar esto, la imagen se encuentra íntegramente en memoria. El siguiente código que se ejecuta es el de *setup.S*, justo en 0x90200. Sigue siendo código real de 16 bits, y vuelve a emplear la BIOS para detectar el hardware instalado (discos, RAM, ratón y tipo de tarjeta de vídeo). Justo después, se reprograman las interrupciones, se construye un descriptor inicial para construir un mapa lógico de memoria (8 Mb para código y 8 Mb para datos, suficiente para el núcleo de Linux tal y como se conoce hoy), y se pasa a modo protegido.

Desde este momento, la BIOS de 16 bits ha quedado fuera de la vista del núcleo, y ya no se volverá a usar. Excepto por el inicio, no es necesaria para Linux. Esto quiere decir que, si no se empleáse DOS con frecuencia, podríamos deshabilitar la opción "*shadow ram*" en el ordenador, ganando hasta 128 Kb de RAM para la ejecución de Linux. (Como todo el mundo sabe, la "shadow" es una copia física del código de ROM en RAM, que es una memoria mas rápida, con lo que optimiza el acceso a funciones de la BIOS, sólo empleadas en DOS).

Algunos usuarios han comentado la idea de grabar Linux en la memoria ROM (flash) de su máquina, para arrancar siempre con este sistema. Los obstáculos que se deberían salvar para realizar esto serían:

1. realizar el diagnóstico inicial (POST) con código propio,
2. detectar el hardware (chipset y periféricos) por acceso directo a los puertos,
3. implementar un "setup" para grabar la CMOS con la configuración de la máquina (número y tipo de discos, disquetes...)

Algunos usuarios han comentado la idea de grabar Linux en la memoria ROM (flash) de su máquina, para arrancar siempre con este sistema

Como esto es complicado y depende de cada máquina, la idea anterior resulta bastante complicada de llevar a la práctica. Linux permanece ligado a la existencia de una BIOS para poder arrancar, puesto que forma parte del estándar del PC, y probablemente este aspecto permanecerá inalterable durante mucho tiempo.

Una vez en modo protegido, el núcleo comprimido, en la dirección 0x10000, debe descomprimirse. El código referente a esto está en *arch/i386/boot/compressed*. En las primeras posiciones se encuentra *head.S*, que contiene ya código de 32 bits, protegido. Chequea que, efectivamente, se encuentra en este modo (si no es así, lo considera un fallo grave, y se queda en bucle), descomprime el núcleo (llamando al código C de este directorio, que es una parte del *gzip*, y cuyo cuerpo principal se encuentra en la rutina "decompress_kernel", en *misc.c*), lo almacena a partir de la dirección definitiva 0x100000, y le da control.



LINUX EN EJECUCIÓN

Tras este pequeño embrollo, se deberían haber quedado claros los rudimentos del inicio de Linux. A partir de aquí, lo que se ejecuta es una imagen descomprimida, justo la que equivale al fichero *vmlinux* (binario situado en */usr/src/linux* tras haber realizado la compilación del núcleo al menos una vez).

Se llega ahora a la parte más interesante para los propósitos. En *init/main.c* se encuentra la rutina principal del núcleo, llamada en el primer momento de ejecución. Aquí se inicializan todos los componentes del sistema, se reserva memoria para los manejadores de dispositivos y estructuras internas, se intenta montar el dispositivo raíz y se da control al primer programa del sistema.

A continuación se verá todo lo anterior con un poco más de detalle.

En primer lugar, se analiza una zona especial, denominada parámetros de arranque, y que contiene los datos que se analizarán en la siguiente sección, apuntados por "command_line". Esto se hace en "start_kernel".

Cuando se termina el análisis de esta zona, se ha obtenido una estructura que se pasará a todas las rutinas de inicialización, para modificar posiblemente su comportamiento. Todas las rutinas de este tipo se llaman, genéricamente, "funcion_init", y el orden de invocación es importante, puesto que unos servicios se van a basar en otros.

El último paso de la inicialización montará un dispositivo como raíz del sistema de ficheros. Si esta operación falla, el núcleo se para, puesto que no va a poder acceder a ningún proceso para comenzar a dar vida al sistema.

Tras la inicialización, se llama a la primitiva "*fork*" por primera vez, lo cual crea una copia del núcleo, que pasa a ejecutar como un proceso independiente, el primero del sistema. Este proceso tendrá una vida muy corta, puesto que la ejecución de la rutina "init" buscará un proceso que ejecutar, y le dará control.

En primer lugar, se intenta abrir el dispositivo "*/dev/tty1*", que es la consola de arranque por defecto. Sin ella no se podría ver ningún mensaje a partir de este punto. Al cumplir este paso, los descriptores *stdin*, *stdout* y *stderr* se inician con valores correctos, y se heredarán en todos los procesos que se ejecuten desde ahora.

Lo siguiente es buscar el programa "*init*", primero en */etc*, luego en */sbin* y por último */bin*. Si se localiza, se le pasa la ejecución, sustituyendo el código del núcleo (el que está como proceso tras el *fork*, no confundirlo con el monolito del núcleo, que siempre se encuentra en memoria) por el de éste programa.

Si no se le encuentra, se intenta un procedimiento de emergencia, buscando la shell */bin/sh*, para darle ejecución. Con esto se da una oportunidad al administrador para que pueda teclear algunos comandos y solucionar el problema. En el peor caso, tocará volver a instalar al menos parte del sistema.

A partir de este instante, acaba la inicialización, y el núcleo pasará a ejecutarse cuando sean invocados sus servicios por parte de los procesos. El comportamiento del programa "*init*" ya ha sido descrito en anteriores ocasiones, y volverá a ser comentado en profundidad en algún artículo posterior.

En definitiva, se ve que lo único que buscará el núcleo antes de ceder la ejecución al primer proceso es:

- Un sistema de ficheros montado en / (dispositivo *root*)
- El dispositivo */dev/tty1*
- El programa */etc/init* ó en su defecto,
- */sbin/init* ó en su defecto,
- */bin/init* ó en su defecto,
- */bin/sh*

Todo este montaje de páginas, asignación de direcciones, descriptores y demás, deben ser construidos en algún momento

PARÁMETROS DE ARRANQUE

De la observación del fichero *init/main.c* se deduce que dentro de la estructura "*bootsetups*" contiene todos los parámetros que el núcleo entiende para inicializarse. La forma genérica de

---

En el número de Mayo de la revista Linux Journal, aparece una sorprendente noticia: La compañia Apple Computer firma un acuerdo estratégico con OSF (Open Software Foundation) para dar soporte a una versión de Linux específica para PowerPC. (*http://www.linuxppc.org/linuxppc/*). OSF migró Linux a OSF MK, un microkernel basado en Mach 3, la versión más moderna de Unix, sobre un ordenador Motorola PowerStack. Tras ello, el portado de este sistema Linux a un PowerMac se realizó en un tiempo récord de tres semanas.

El soporte de Linux por parte de Apple, bastante sorprendente teniendo en cuenta la política seguida hasta ahora, se toma dentro de la compañía como "una cuestión estratégica", y se repite así el camino ya tomado por DEC en el mismo sentido para su plataforma Alpha, hace un año.

El portado de Linux por parte de la organización OSF abre nuevas posibilidades para el sistema operativo, ya que la plataforma OSF MK ha sido portada a arquitecturas originalmente fuera del alcance de Linux, como el Intel y860 y la arquitectura PA-RISC de HP. (recordemos que el Intel P7 "Merced" es un PA-RISC, y su lanzamiento ocurrirá a finales de este año).

Aunque el "OSF MK Linux server" diverge del diseño original de Linux como un sistema monolítico, frente a la arquitectura en "hojas de cebolla" del microkernel, en realidad sólo han sido modificados 43 de los 909 ficheros que forman el núcleo de Linux. Se puede obtener más información en *http://www.osf.org/mall/os/mklinux.html*

Cuadro 1: El acuerdo entre Apple y OSF.

para realizar la introducción de parámetros es

nombre=dato1,dato2,dato3,...,datoN

No debe haber espacios aquí, y se admiten hasta 10 datos. Éstos son números decimales, o hexadecimales si comienzan con el prefijo "0x". El valor de "nombre" debe estar reconocido en la estructura "bootsetups", y se muestra un listado de las posibilidades en el cuadro 2.

Si los datos no se reconocen como números, la rutina de inicialización específica referenciada en cada parámetro es responsable de analizarlos. Un ejemplo es "floppy=", que admite variaciones tales como:

floppy=<addr>,two_fdc
floppy=<n_drive>,<tipo>,cmos
floppy=thinkpad

Linux se autodescomprime en arranque a partir de una imagen de hasta 508Kb

El primer ejemplo sirve para que Linux reconozca un segundo controlador de disquetes (ya que la BIOS sólo permite uno), en el puerto *<addr>*. Con ello se tendría hasta cuatro disqueteras, tal y como admite el estándar del PC.

El segundo indica el tipo de unidad para las disqueteras "extras". El campo *<n_drive>* varía entre 0 y 3, y el *<tipo>* entre 1 y 6, siendo, sucesivamente, unidades de 360K, 1.2M, 720K, 1.44M, cinta y 2.88M.

El tercero indica que la unidad es la de un modelo específico de portátil de IBM, que sigue un método no compatible para sus unidades. Los lectores interesados en todos los detalles de este aspecto, se pueden encontrar en *drivers/block/README.fd*.

Por supuesto, es posible poner varios parámetros para el núcleo, separados por un espacio. Como ejemplo ilustrativo, supóngase que se tiene una tercera unidad de floppy, que es de 2.88M, con la controladora en el puerto 0x2C0. Lo que se debería poner como parámetros de arranque es:

floppy=0x2C0,
two_fdc floppy=2,6,cmos

Hay varias formas de indicar estos parámetros:

- Con LOADLIN, se indican en la misma línea de comando en la que se le invoca.
- Con arranque con floppy, aparece una línea de prompt (el conocido "boot:" que nos encontramos al realizar nuestra instalación) equivalente a la de LILO en ejecución.

Se pueden introducir parámetros de arranque justo antes de la ejecución del sistema

- Con LILO, en configuración, se puede indicar la línea *append = "..."*, poniendo las opciones de arranque entre las dobles comillas.
- Con LILO, en ejecución, cuando se pulsa una tecla para que aparezca el prompt, es posible elegir una imagen de núcleo y poner parámetros adicionales.

En el último ejemplo, suponiendo que la imagen de núcleo a cargar sea "*minucleo*", la indicación de que el root es la primera partición del primer disco IDE sería:

minucleo root=/dev/hda1

Mediante la tabla adjunta, los lectores que tuviesen problemas con la detección de su hardware, pueden probar con los parámetros detallados allí. En muchos casos, como el lector CDU31A de SONY, o una segunda tarjeta de ethernet, es lo único necesario para un arranque exitoso. En otros casos, como ocurre con los poseedores de una tarjeta ne2000 compatible, se debe combinar esto con un programa de configuración de la EEPROM de la tarjeta.

De nuevo, se hace hincapié en la utilidad de revisar, al menos ligeramente, el núcleo. La lista de la tabla, tan necesaria para los que no tienen un hardware que detecte automáticamente Linux, se obtuvo tras una observación de apenas un par de horas, sin conocimiento previo del contenido de los ficheros consultados.

Un último ejemplo, útil para los que tengan un CD IDE/ATAPI como maestro en la controladora secundaria (que no es detectada por defecto en Linux) es tan sencillo como:

hdc=cdrom

CONCLUSIONES

En este número se ha dado un pequeño repaso por los primeros instantes de la ejecución del sistema preferido. Con ésta, se comienza un pequeño viaje por la estructura del núcleo, en posteriores entregas, aunque intercalados con otros temas de "no tan bajo nivel", para que ningún lector se aburra.

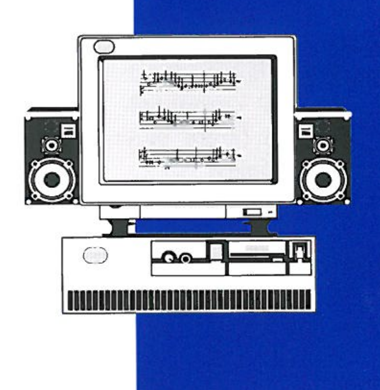
Esto puede ser útil para explicar comportamientos "extraños", sobre todo en el arranque, debido a la instalación de nuestro hardware. En este número, por ejemplo, se ofrece como consecuencia del análisis de las fuentes, todos los parámetros del arranque de la versión 1.2.13 de Linux.

También se ha visto una pequeña introducción al modelo de memoria de Linux, para el entorno multitarea que ofrece, que ayudará a comprender el funcionamiento del sistema.

En el próximo mes se verá cómo modificar el ramdisk de Linux para que sirva como raíz del sistema de ficheros (Esta habilidad aparece ya en los núcleos 1.3.8x). Como esto se realiza en la propia imagen de núcleo, esto puede servir para construir un arranque "de emergencia", con todas las utilidades en un disquette. Hasta la próxima.

FICHEROS VOC

Agustín Guillén



Se continúa con la serie sobre ficheros de formatos de sonido. En esta ocasión el artículo va a tratar sobre ficheros que almacenan sonido propiamente dicho. Se componen de secuencias de datos, que una vez convertidas de su formato binario (digital) a uno de impulsos de magnitud variable (analógico) se vuelve a obtener el sonido original. En la conversión que los creó (de sonido analógico a datos digitales) se perdió calidad, pero con las altas velocidades de digitalización actuales (sobre los 44 Kkz) y con la resolución de los datos (16 bits en casi todas las tarjetas modernas), la perdida casi no es apreciable por el oído humano. La conversión entre los distintos formatos de ficheros de sonidos *sampleados* suele ser muy sencilla, pues basta con aislar la parte que contiene los datos propiamente dichos y crear una cabecera para el nuevo formato. En ocasiones, también es necesario el convertir los datos, pues se pueden utilizar amplitudes distintas para el rango de valores y oscilar entre 256 valores diferentes (de -128 a 128) o existir 65.536 variaciones (de -32.768 a 32.768). En cualquier caso, bastará con multiplicar o dividir cada dato, por la correspondiente diferencia de "sensibilidad", de la siguiente forma:

Para convertir datos de 256 valores a 65.536 valores:
65.536 / 256 = 256

Habrá que multiplicar cada dato de sonido por 256, para ajustarlo a la nueva escala.

Nótese que en las conversiones de los datos almacenados en sensibilidades altas a datos de escala más reducida, se perderá calidad de detalle.

Los formatos de ficheros más utilizados para almacenar sonidos digitalizados, junto con su plataforma habitual, son los siguientes:

- Sound Blaster (*.VOC)
- Sun, NeXT, DEC (*.AU)
- Windows (*.WAV)
- Sounder, Soundtools (*.SND)
- Amiga (*.8SVX y *.IFF)

FORMATO

Los creadores de la extendida tarjeta de sonido Sound Blaster, crearon casi simultáneamente el formato VOC, denominado *Creative Voice Format*. Su formato se diseñó para poder reproducirse casi directamente en las tarjetas recién creadas. Deseaban (y consiguieron) que con sólo leer un fichero en memoria, y cargar un registro *hardware* de la tarjeta con un puntero a los datos, la reproducción comenzara.

Los ficheros VOC se componen de dos bloques principales: el bloque de cabecera y el bloque de datos.

La cabecera guarda los valores necesarios para identificar al fichero (identificador del fichero, versión, etc.)

Digitalizando sonidos a 44 KHz de muestreo no existen pérdidas apreciables por el oído humano

Los sonidos se pueden *samplear* o digitalizar mediante módulos ADC (Conversores Analógico a Digital), creando ficheros con los datos muestreados en esa digitalización. A estos datos puros o *raw* se les añade unas cabeceras descriptivas y se obtienen los ficheros VOC o WAV

y su formato puede observarse en la figura 1. Un ejemplo de cabecera sería:

"Creative Voice File", 1AH
Cadena identificativa
1AH Offset a datos
010AH Versión 1.10
1129H Complemento de 010AH + 1234H

La otra parte del fichero, el bloque de datos, puede almacenar varios sub-bloques de distintos tipos. A continuación se listan los tipos de sub-bloques con su descripción, figurando debajo los campos que los componen con su tamaño en bytes (entre paréntesis) y sus valores habituales:

Cabecera de los ficheros VOC			
Nombre	**Tamaño**	**Tipo de datos**	**Descripción**
Nombre identificativo	20	char	Debe aparecer la cadena "Creative Voice File",1AH
Offset de los datos	2	unsigned short	Puntero al bloque de datos del fichero, usualmente aparece el valor 1AH
Versión del VOC	2	unsigned short	En el byte LSB (byte bajo o *Less Significant Byte*) está el número menor de versión y en el MSB (byte alto o *More...*) el número mayor
Identificador	2	unsigned short	Número complementario del número de versión + 1234H, se trata de una especie de protección contra errores

Los creadores de la Sound Blaster, crearon casi simultáneamente el formato VOC

TIPO 0.- FIN DEL BLOQUE DE DATOS

Tipo (1)

Debe ser el último bloque del fichero e indica que no hay más bloques disponibles.

TIPO 1.- DATOS DE SONIDO

Tipo (1), Longitud (3), Velocidad de muestreo (1), Método de compresión (1), Datos (Longitud - 2)

Marca el inicio de un nuevo bloque de datos de sonido. En el campo *Velocidad de muestreo* se define la velocidad correcta de reproducción y su formato es el siguiente:

Velocidad de muestreo indicada = 256 - (1000000 / Velocidad de muestreo real)

De esta forma, basándose en esta fórmula, si se desea una velocidad de muestreo de 20 Khz, será necesario realizar la siguiente operación, con lo que se obtendrá:

256 - (1000000 / 20000) = 206

Por lo tanto, éste será el valor que se deberá coger para indicarlo en el campo *Velocidad de muestreo.*

Los distintos tipos de compresión permitidos son:

0.- 8 bits sin compresión
1.- compresión 4 bits
2.- compresión 2.6 bits
3.- compresión 2 bits
4.- 1 canal multi
5.- 2 canal multi
6.- 3 canal multi
7.- 4 canal multi
8.- 5 canal multi
9.- 6 canal multi
10.- 7 canal multi

Además, es necesario tener en cuenta que cada bloque de este tipo puede tener una velocidad diferente de muestreo o un método diferente de compresión.

TIPO 2.- CONTINUACIÓN DE LOS DATOS DE SONIDO

Tipo (1) , Longitud (3), Datos (Longitud)

Continúa con los datos de sonido desde el último bloque del tipo 1 (Datos de sonido).

TIPO 3.- SILENCIO

Tipo (1) , Longitud (3), Retardo (2), Velocidad de muestreo (1)

Define un periodo de silencio en la reproducción del sonido. El retardo está definido en unidades de la velocidad de muestreo indicada en el mismo bloque.

TIPO 4.- MARCADOR

Tipo (1) , Longitud (3), Marcador (2)

Este bloque está pensado para sincronizar cualquier evento externo con el sonido del fichero. Al encontrarse este bloque, el programa reproductor del sonido carga una variable con el valor del campo *Marcador* y de esta manera se pueden ejecutar procesos en puntos determinados de la reproducción. Esto es muy útil para demostraciones o juegos, en los que se tiene que animar algún objeto conjuntamente con el sonido. Los valores *0H* y *FFFFH* están reservados por Creative Labs.

TIPO 5.- TEXTO ASCII

Tipo (1) , Longitud (3), Cadena ASCII (Longitud), Fin de cadena NULL (1)

Este tipo de subçbloque almacena una cadena ASCII pensada para guardar comentarios o nombres referidos a los sonidos incluidos en el fichero.

Con los sub-bloques de tipo 8 se consigue una mayor velocidad de muestreo

TIPO 6.- BUCLE REPETICIÓN

Tipo (1) , Longitud (3), Contador (2)

Indica el inicio de un bucle de repetición. Los bloques de datos que se encuentren entre este bloque y el siguiente bloque del tipo 7 (Fin bucle repetición) serán repetidos *Contador + 1* veces.

TIPO 7.- FIN BUCLE REPETICIÓN

Tipo (1) , Longitud (3)

Mediante este bloque se marca el final del bucle de repetición.

Los ficheros VOC se componen de dos tipos de bloques : de Cabecera y de Datos

TIPO 8.- EXTENSIÓN

Tipo (1), Longitud (3), Velocidad de muestreo (2), Método de compresión (1), Modo de reproducción (1), Datos (Longitud - 2)

Se trata de un bloque similar a los de tipo 1 (Datos de sonido), pero pensado para adaptarse a las modernas especificaciones que presentan las tarjetas con las que se trabaja en la actualidad. La utilización de este tipo de bloque permite alcanzar una mayor velocidad de muestreo al indicarse dicho valor mediante dos bytes. Además, también especifica el tipo de reproducción (estéreo o mono), debido a que las tarjetas de sonido iniciales sólo reproducían sonidos de un sólo canal o *mono* (1 solo DAC).

El bloque de datos puede almacenar sub-bloques de distintos tipos

En este caso la velocidad de muestreo se obtiene mediante una forma distinta. Esta nueva forma consiste en la aplicación de la siguiente fórmula a los datos almacenados:

Velocidad de muestreo indicada = 65536 -
(256000000 / Vel. de muestreo real)

Si se quiere indicar una velocidad de muestreo de 40 Khz, se obtendrá: 65536 - (256000000 / 40000) = 59136, valor a colocar en el campo *Velocidad de muestreo*.

REPRODUCCIÓN

La reproducción de los ficheros VOC simples (y sin compresión) no es complicada y mediante un pequeño programa se pueden conseguir buenos resultados. En el disco que acompaña este número de la revista se ha incluido un programa, desarrollado en C, que reproduce ficheros VOC, aunque es bastante legible por neófitos en este lenguaje. Los pasos a seguir para la reproducción de un fichero VOC son los siguientes:

- Definir una estructura de datos para poder albergar la cabecera de manera cómoda y directa.
- Leer la cabecera del programa y comprobar exhaustivamente que se trata de un fichero VOC (los resultados de abrir otro tipo de fichero y buscar los bloques de datos en su interior son imprescindibles). Para la comprobación, basta con comprobar la cadena habitual ("Creative Voice File") con su byte de terminación (1AH), pero es preferible comprobar también el código de identificación del fichero mediante el algoritmo indicado en el fuente.
- Ir leyendo los bloques contenidos en el fichero, hasta encontrar un bloque del tipo 0 (Terminador), que indique el final del fichero. Según el tipo de bloque encontrado, se actuará de una forma u otra. Lo habitual es encontrar un sólo bloque del tipo 1 (Datos de sonido) en el que se incluyen las muestras del sonido a reproducir. En el ejemplo sólo se tienen en cuenta los bloques de datos sin compresión, de cara a clarificarlo un poco. Una vez localizados los datos o *samples* sólo resta activar la tarjeta de sonido y enviarlos al DAC, ya sea dato a dato o mediante una transferencia DMA. Si se elige este último método, se puede optar por esperar a que la transferencia termine, o continuar realizando cualquier otra actividad, comprobando de vez en cuando el contador DMA para conocer si el proceso terminó.

Para contactar con el autor :

Fidonet 2:341/43.7
Internet aguillen@abcnet.es

PROGRAMAMCIÓN DEL COPROCESADOR 387

Enrique de Alarcón

Gracias al coprocesador matemático, es posible disponer en los programas escritos en ensamblador, de todas las posibilidades matemáticas de una calculadora científica, pudiendo realizar operaciones sin necesidad de usar ninguna clase de librería de emulación, lo cual ayuda mucho, sobre todo si se va a crear aplicaciones CAD y de renderizado, que requieren mucha potencia de cálculo.

Hasta la aparición del microprocesador 80386, los usuarios prácticamente ni sabían de la existencia de un coprocesador matemático llamado 8087/287/387 que se podía instalar en algunos casos en la placa base y que podía realizar todo tipo de cálculos matemáticos usando números reales como si de una calculadora científica se tratara.

Ello se destino en inicio como un complemento hardware caro y que sólo necesitaban los usuarios que requerían una gran capacidad de cálculo en sus sistemas, como es el caso de arquitectos o matemáticos, que usan programas de alto consumo matemático.

Más adelante, al aparecer el 486, que en muchos modelos, ya incluía el 487 en el mismo chip (que es un 387 algo mejorado) y debido también a la existencia de un mayor número de aplicaciones científicas y programas de renderizado al alcance del gran público que usaban de sus servicios para funcionar hasta un 200% más rápido, su uso se popularizó mucho más.

Actualmente, prácticamente todos los programadores, y gran parte de los usuarios, saben de su existencia, y su utilidad se ha visto confirmada en los micros de última generación como son Pentium y Pentium Pro, donde el coprocesador viene ya en todos los modelos sin excepción.

En este artículo se va a explicar cómo se puede programar dicho coprocesador, pudiendo así disfrutar de sus valiosas posibilidades de cálculo en los programas.

FUNCIONAMIENTO DEL 387

El coprocesador tiene un conjunto de instrucciones propio, al igual que lo tiene el 386, mediante las cuales se puede programar.

Tanto para intercambiar datos con un programa, para gestión de errores y para su control y estado, posee tres conjuntos de registros propios que son:

1.- Un conjunto de 8 registros (de 80 bits cada uno) donde se almacenan datos, a los que se accede como si de una pila de tratara. La pila se llama *ST* y los elementos que se van almacenando se llaman *ST(num)*. Por ejemplo, si se almacena el valor 5.1 y el valor 3.2 (en éste orden), *ST(1)* tendrá el valor 5.1 y *ST(0)* contendrá el 3.2. Normalmente, *ST(0)* se usa para almacenar un parámetro para la instrucción de cálculo que se desea ejecutar posteriormente.

2.- Cuatro registros (de 32 bits cada uno) donde se almacenan todos los datos relacionados con el último error producido. Debe recordarse que, dependiendo de si se está en modo real o modo protegido, se dirá que dichos punteros contienen segmentos o selectores y offsets de 16 o 32 bits respectivamente. Dichos registros son los siguientes:

- *FIP*: Contiene el offset a la instrucción 387 que ha producido el error.
- *FCS*: Contiene el selector donde está la instrucción 387 que ha dado error.
- *FOO*: Offset al operador de memoria usado por la instrucción que dio error.
- *FOS*: Selector del operador de memoria.

3.- Tres registros de estado y control (de 16 bits cada uno) donde hay toda clase de información de funcionamiento y estado del 387.

REGISTROS DE ESTADO Y CONTROL

Estos tres registros de 16 bits cada uno, son: La *palabra de marcas* o *tags*, la *palabra de control* y la *palabra de estado*.

LA PALABRA DE MARCAS O TW:

Este registro contiene ocho campos de 2 bits, cada uno los cuales corresponde a cada uno de los ocho registros físicos de la pila. Cada campo de 2 bits contiene el estado de su contenido y de este modo también es posible detectar desbordamientos superiores e inferiores de la pila. Por ejemplo, para detectar un desbordamiento por arriba al almacenar un dato en la pila, el coprocesador sólo tiene que mirar si el puntero a pila (llamado *TOP*) apunta después de ser incrementado a un registro que ya contiene algo, con lo que dará a conocer que se ha dado toda la vuelta a la pila y se ha *entrado* por abajo. El caso de desbordamiento inferior es lo contrario, detectar si se pasa a apuntar a un registro que no contiene nada.

Como se mencionado anteriormente, cada campo posee 2 bits de datos, con lo que se tienen 4 valores posibles de información por registro de pila. Si el campo posee el valor *00b*, significa que contiene un número válido. Si es *01b* significa que su contenido es cero. Si es *10b* es contiene un valor especial, como puede ser, por ejemplo, un valor infinito. Por último, si posee el valor *11b*, indica que el registro de pila no posee ningún valor, está vacío.

LA PALABRA DE CONTROL O CW:

Esta palabra contiene varios campos modificables por el programador, donde se indica al 387 cómo debe funcionar en ciertas situaciones. A continuación se explican todos los campos que posee, empezando por los bit de derecha a izquierda:

- Bit 0, *IM*: Máscara de excepción para error de *operación no válida*.
- Bit 1, *DM*: Máscara de excepción para error de *no normalizado*.
- Bit 2, *ZM*: Máscara de excepción para error de *División por cero*.
- Bit 3, *OM*: Máscara de excepción para error de *Desbordamiento*.
- Bit 4, *UM*: Máscara de error *Desbordamiento inferior*.
- Bit 5, *PM*: Máscara de error de *Precisión*.
- Bits 8-9, *PC*: *Control de precisión*, los resultados de las operaciones que realiza el 387 en punto flotante se redondean con uno de los 3 siguientes métodos de precisión. Si el campo *PC* contiene 00b la precisión de la mantisa será de 24 bits, que corresponde a un real corto. Si contiene el valor 01b significa reservado. Si contiene 10b, a la mantisa se le asignan 53 bits de precisión, que corresponde a un real largo. Por último, si el campo contiene 11b, se dan 64 bits a la mantisa para dar un real temporal.
- Bits 10-11, *RC*: *Control de redondeo*, donde hay cuatro posibilidades a la hora de redondear resultados. Si el campo *RC* contiene el valor 00b el valor se redondea hacia el más cercano o al par. Si contiene el valor 01b se redondea hacia -infinito. Si contiene 10b se redondea hacia +infinito. Por último, si contiene 11b, se trunca hacia 0.
- Bits 12-15: Estos cuatro últimos bits forman parte de los campos de definición de redondeos y precisión, pero aún no están definidos en el 387 (libres próximas generaciones de procesadores), lo más recomendable es que no se altere su contenido nunca para evitar la perdida de compatibilidad de los programas con los futuros chips matemáticos que usen dichos flags.

LA PALABRA DE ESTADO O SW:

Su contenido puede ser examinado para poder detectar situaciones especiales. A continuación se detallan todos los campos que incluye, de derecha a izquierda:

- Bits 0-5, *IE-DE-ZE-OE-UE-PE*: Corresponden a cada una de las seis posibles excepciones de error que se indicaban en la palabra de control. Se ponen a 1 si se produce su correspondiente error. Estos sólo se ponen a 0 si lo hace el propio programador.
- Bit 6, *IF*: Indicador de fallo de pila. Si está a 1, indica que se ha producido un error de desbordamiento ya sea por arriba o por abajo.
- Bit 7, *IR*: Este Bit esta a 1 si se ha producido una excepción de error enmascarado.
- Bit 8, *C0*: Flag usado en saltos condicionales (equivalente a banderas condición en 386).
- Bit 9, *C1*: Flag usado en saltos condicionales.
- Bit 10, *C2*: Flag usado en saltos condicionales.
- Bits 11-13, *TOP*: es el puntero *TOP*, donde se indica a donde apunta el puntero dentro de la pila.
- Bit 15, *B*: Funciona junta con el bit 7, para control de errores no enmascarados.

El 387 posee un set de instrucciones propio formado por 54 funciones

TIPOS DE DATOS DEL 387 , EL IEEE

El coprocesador soporta tanto los modelos de datos del 386, enteros (excepto los de 8 bits) y BCD como los de punto flotante, los cuales son específicos del 387 y no son más que números reales almacenados de un modo especial que les da gran precisión y un gran rango de contenido a la vez.

Hace muchos años, cada tipo de microprocesador, usaba un tipo propio de almacenamiento de los datos de punto flotante, y al no haber un estándar, una misma aplicación, ejecutada en diferentes ordenadores, daba resultados diferentes.

Con vistas al futuro, y antes de crear ningún tipo de procesador de que usara cálculos con punto flotante, Intel incitó al IEEE a que creara un estándar para este tipo de datos, el cual se adoptó en el 8087.

Este estándar sería el llamado *Standar for Binary Floating-Point Arithmetic* (Estándar para aritmética de Punto flotante) y debido a su buen diseño, daba varias ventajas que a grandes rasgos eran: Simplificar la conversión de programas de cálculo entre diferentes modelos de máquinas y facilitar la creación de aplicaciones científicas sin necesidad de muchos conocimientos en métodos numéricos.

La aceptación de este estándar ha sido buena, y todos los procesadores Intel posteriores lo han usado desde entonces, además de muchos otros modelos de micros que también se han añadido al uso del mismo. A grandes rasgos, los números de punto flotante usados por Intel, están formados siempre por una mantisa (o base), un exponente, y un bit de signo.

Los tipos de datos soportados por el 387 son:

1. *Entero palabra*: 16 bits, con un rango de -32768 hasta 32767.
2. *Entero corto*: 32 bits, rango desde -2*10^9 hasta +2*10^9.
3. *Entero largo*: 64 bits, rango -9*10^18 hasta +9*10^18.
4. *Decimal Empaquetado BCD*: 80 bits, -(8 dígitos de tipo <9>) hasta +(8 dígitos del tipo <9>).
5. *Real corto*: 32 bits, -3.39*10^(-38) hasta 3.39*10^38.
6. *Real largo*: 64 bits, -1.80*10^(-308) hasta +1.80*10^308.
7. *Real Temporal*: -1.19*10^(-4932) hasta +1.1º9*10^4932).

TIPOS DE DATOS ESPECIALES

Dentro del los tres tipos de datos de punto flotante (reales de 32, 64 y 80 bits), que son los tipos 5-7 de la lista anterior, hay varios tipos de representación para los datos según el contenido que se quiere almacenar. A continuación se detallan todos:

1. *Ceros*: Es un caso donde simplemente se ha desplazado el exponente en 0 con una mantisa de valor 0.
2. Infinitos: Para representar los infinitos, se pone todos los bits del exponente a 1 (o sea, al valor máximo) con la mantisa con valor 0.
3. *No normalizados*: Es un sistema para poder representar números muy pequeños perdiendo precisión gradualmente y no de golpe, como lo hacen otros micros. En otros sistemas, cuando se sobrepasa el número más pequeño posible representable, se pasa directamente al cero, lo cual da una perdida de precisión muy brusca.
4. *Pseudo-no normalizados*: Este tipo de datos no puede ser generado por el 387, pero sí que puede usarlo. Es un tipo de dato en desuso debido a que tiene formas normalizadas equivalentes para ser representado.
5. *Tipo NaN*: que poseen dos variantes, *NaN señalizadores* y *silenciosos*. *NaN* significa *Not a Number*.
6. *Los indefinidos*: Estos números, que son un tipo especial de *NaN silencioso*, se obtienen al producirse una excepción de operación no válida y cuando ningún operando de la instrucción era un *Nan*.

También hay una serie métodos de representación que no son aceptadas por el 387, pero que se usaban en los 287. Estos no soportados son: *Pseudo-Nan, Pseudo-Infinitos, No normalizados* y *Pseudo-ceros*.

SET DE INSTRUCCIONES

El coprocesador dispone de 54 instrucciones, las cuales, en su gran mayoría, están destinadas a realizar operaciones matemáticas. Todas las instrucciones del 387 empiezan por la letra "F", y seguidamente contienen una abreviación de unas pocas letras con la tarea que realizan. Así, por ejemplo, la instrucción del 387 destinada a sumar dos números, se llama *"F" + "ADD"*, es decir, *FADD*, y así con todas las instrucciones.

Las instrucciones que pueden operar con reales o con enteros se diferencian en que las segundas añaden la letra "I" de *integer* después de la "F". Así pues, sumar un real al contenido de la pila se llamará *FADD*, y sumar un entero a la cima se hará con *FIADD*.

Aquí está la lista de todas las instrucciones y su funcionalidad:

- *F2XM1*: Calcula el valor (2^X -1).
- *ABS*: Valor absoluto.
- *FADD/FIADD/FADDP*: Suma de dos números.
- *FBLD*: Carga número BCD en pila.
- *FBLDP*: Se convierte *ST(0)*, la pila de la cima a un número BCD y se extrae de la pila.
- *FCHS*: Cambia el signo.
- *FCLEX/FNCLEX*: Borra excepciones.
- *FCOM/FCOMP/FCOMPP*: Compara dos números.
- *FCOS*: Calcula en coseno.
- *FDECSTP*: Decrementa puntero a pila.
- *FDIV/FIDIV/FDIVP*: Divide.
- *FDIVR/FIDIVR/FIDIVRP*: Realiza la división inversa.
- *FFREE*: Libera valor de la cima de la pila.
- *FICOM/FICOMP*: Compara dos enteros.
- *FILD*: Carga entero en la pila.
- *FINCSTP*: Incrementa puntero la pila.
- *FINIT/FNINIT*: Resetea el coprocesador.
- *FIST*: extrae el valor de cima de la pila. (sería como leer sólo la cima de la pila con *MOV AX,SS:SP*)
- *FISTP*: Extrae el valor de *ST(0)* de la pila y lo libera de la misma (sería como un *POP* en la pila normal).
- *FLD*: Carga un real en la pila.

El 387 tiene tres conjuntos de registros, de datos, de errores y de estado/control

- *FLDcon*: Carga una constante, donde *CON* puede ser: 1 (1.0) ,L2E (log2e), L2T (log2 10), LG2 (log10 2), LN2 (loge2), PI (3.14...), Z (0). Así, por ejemplo, *FLDPI*, carga en cima de la pila el valor de PI.
- *FLDCW*: Carga palabra de control.
- *FLDENV*: Carga entorno.
- *FMUL/FIMUL/FMULP*: Multiplica dos números.
- *FNOP*: No operación.
- *FPATAN*: Calcula Arcotangente parcial.
- *FPREM*: Calcula el resto parcial.
- *FPPREM1*: Calcula resto parcial según IEEE.
- *FPTAN*: Calcula tangente parcial.
- *FRNDINT*: Redondea resultado hacia entero.
- *FRSTOR*: Restaura estado.
- *FSAVE/FNSAVE*: Almacena estado.
- *FSCALE*: Calcula escalas de potencias de 2.

- *FSIN*: Calcula seno.
- *FSINCOS*: Calcula seno y coseno.
- *FSQRT*: Calcula la raíz cuadrada.
- *FST*: Extrae un real de la pila.
- *FSTCW/FNSTCW*: Almacena la palabra de control.
- *FSTENV/FNSTENV*: Almacena entorno.
- *FSTP*: Extrae un real de la pila y lo libera de la misma.
- *FSTSW/FNSTSW*: Extrae la palabra de estado.

Los ocho registros donde se almacenan los datos, funcionan como una pila

- *FSTSW AX/FNSTSW AX*: Extrae palabra de estado y la deposita en AX.
- *FSUB/FISUB/FSUBP*: Resta dos números.
- *FSUBR/FISUBR/FSUBRP*: Resta inversa de dos números.
- *FTST*: Comprueba.
- *FUCOM/FUCOMP/FUCOMPP*: Compara dos números de forma desordenada.
- *FXAM*: Examina.
- *FXCH*: Intercambia dos valores de pila.
- *FXTRACT*: Extrae el exponente y la mantisa de *ST(0)*.
- *FYL2X*: Calcula (*Y * log2 X*).
- *FYL2XP1*: Calcula (*Y*(X+1)*).
- *WAIT*: Espera al coprocesador que acabe de procesar. Sirve para sincronizar el 386 y el 387.

DÓNDE PONER EL CÓDIGO 387

Cuando se escribe un programa que usa instrucciones de 387, es posible escribirlas donde más convenga, insertándolas entre medio de las instrucciones de código del 386 que se desean y mezclando así instrucciones de ambos micros. Al resultado se le llama un programa híbrido, y es el propio 386 quien, a medida que va ejecutando el código secuencialmente, va ejecutando sus propias instrucciones e ingorando las de coma flotante, que son captadas y ejecutadas por el 387 automáticamente.

Como se puede deducir, es posible hacer funcionar a los dos micros en paralelo, y hacer que calcule el 387 un coseno mientras el 386 esta ejecutando código propio a parte.

EJEMPLOS DE USO

A continuación se exponen unos cuantos ejemplos de uso para su mejor comprensión.

Ejemplo 1: Suma de dos valores reales de 32 bits y extraer el resultado de la misma. Pasos a realizar: Almacenar los dos valores, realizar la suma, extraer el valor. El código quedaría como sigue:

```
FLD dword ptr ds:(si )
FADD dword ptr di:(si+4)
FSTP dword ptr di:(si+8)
```

En el ejemplo se inserta el primer valor real en la pila, y se le suma el segundo, indicando al mismo directamente como parámetro de la instrucción *FADD*, resaltando así que no es necesario insertar ambos valores en la pila para calcular la suma, aunque sólo se puede poner uno como parámetro directo, ya que las instrucciones del 387 no pueden llevar más de un operando de memoria. *FSTP* extrae el valor ST(0), libera la cima y almacena el valor en la dirección de memoria indicada. Tan solo con cambiar *FSTP* por *FISTP*, el resultado obtenido hubiese sido el número entero de 32 bits correspondiente al resultado, con lo que se pone de manifiesto la facilidad que da el 387 para convertir datos entre diferentes formatos.

Algunos tipos de datos del 287 dejan de usarse en el 387

Otra forma de realizar la misma suma hubiese sido:

```
FLD dword ptr ds:(si)
FLD dword ptr ds:(si+4)
FADDP
FSTP dword ptr ds:(si+8)
```

con la única diferencia que se han almacenado los dos valores en la pila antes de operar sobre ellos. *FADDP* suma ST(0) y ST(1), elimina de la pila a ST(0) y guarda el resultado en ST(1), que ahora pasa a ser ST(0). *FSTP* extrae el nuevo ST(0) y lo elimina de la pila, dejando así a la misma limpia de residuos.

Ejemplo 2: Calcular el seno del ángulo X radianes. Pasos: Almacenar el ángulo (real 32 bits), calcular el seno, extraer resultados y limpiar residuos. Resaltar que, tanto la operación de SENO como COSENO, almacenan también como resultado en la pila el valor 1.0 en ST(0), quedando el valor que interesa en ST(1).

```
FLD dword tpr ds:(si)
FSIN
FINCSTP
FLD dword ptr ds:(si+4)
```

En el ejemplo, *FINCSTP* hace saltar el valor 1.0 de ST(0) que no interesa.

Ejemplo 3: Multiplicar un real por un entero, y sacar el resultado como real. Pasos: Almacenar el real, multiplicarlo por el entero directamente como parámetro y extraer el real resultante.

```
FLD dword ptr ds:(si).
FIMUL dword ptr(si+4).
FSTP dword ptr ds:(si+8)
```

Ejemplo 4: Comparar dos elementos reales de la pila y realizar un salto condicional. Pasos: Almacenar los dos reales, comparar los dos elementos, extraer la palabra de estado del 387 donde están los flags resultantes de la comparación y almacenarla en AX, limpiar pila de residuos y saltar según resultado.

```
FLD ds:(si)
FLD ds:(si+4)
FCOMP ST(1)
FSTSW AX
FINCSTP
FINCSTP
```

SAHF
jp ES_ERROR
je ES_IGUALES
jb ES_MENOR
ja ES_MAYOR

La instrucción *SAHF* es del 386 y lo que hace es almacenar los flags de AH en el registro de flags del 386 para poder ser detectados por las instrucciones de salto, que también son del 386.

Ejemplo 5: Detectar qué modelo de coprocesador está presente en el sistema. Pasos a realizar: Realizar un reset del coprocesador por software con un *FINIT*, leer entorno 287/387 con *FSAVE* apuntando a un buffer libre de 108 bytes donde guardar el entorno, comprobar si los Bit *IS* y *ES* de la palabra de estado están a 1 y si el bit *IM* de la palabra de control está a 0, en cuyo caso significa que hay instalado un 387. En caso contrario, se trata de un 287.

Recordar cómo se almacenan los registros en el buffer: *CW,SW,TW, FIP, FCS, FOO, FOS* y la pila de 8 registros de 80 bits cada uno.

DETALLES TÉCNICOS

Cuando se está habituado a la programación del 387, se da cuenta de que una vez un valor es almacenado en la, ya no se necesita recordar qué tipo de dato contenía dicho registro de la pila (es decir, si se trataba de un real, entero o BCD). Esto es debido a que todos los valores que se introducen a la pila, son convertidos antes por el 387 al tipo *real temporal* (tipo 7 de datos que puede usar) que son números de 80 bits, con capacidad para contener cualquiera de los otros 6 tipos de datos que soporta. Esto lo realiza para que se simplifique el funcionamiento interno de cálculo sobre los valores, pues así sólo tiene que ser capaz de operar realmente sobre un tipo de datos y, además, hace que los valores temporales de las operaciones intermedias que se obtienen con registros de la pila pierdan la menor precisión posible. A si mismo, cuando se realiza una operación indicando un operando directo de memoria que no sea un real temporal, es convertido siempre a dicho tipo antes de trabajar con el mismo.

En la Tw o palabra de marcas está registrado el contenido de cada elemento de la pila

Su mayor virtud es la facilidad para operar con diferentes tipos de datos

ERRORES COMUNES

El hecho de que sea tan fácil operar sobre tipos tan diferentes de datos hace también que un error de programación pueda dar resultados numéricos también muy desastrosos. Así, por ejemplo, que se cargue en la pila un entero con la instrucción *FLD* (y no con *FILD* como debería ser) hará que el 387 almacene dicho valor con un contenido totalmente diferente al que se desea y, por lo tanto, las operaciones que se realicen usando dicho valor darán resultados erróneos.

También hay que tener presente que al funcionar los 8 registros de almacenamiento como una pila, lleva consigo los típicos problemas, como son que se puede tener errores de desbordamiento de pila tanto por arriba como por abajo, y de ahí el énfasis puesto en los ejemplos a la hora de limpiar a la pila de posibles residuos.

Por último, recordar que en el 386 la sincronización entre ambos micros se hace automáticamente, pero que en el 8087, se debe tener en cuenta usar instrucciones *WAIT* para esperar los resultados del coprocesador, pues sino, es posible dejar la instrucción a medio procesar y recoger resultados numéricos que aun no existen. Éste es un detalle que aparece también en el apartado de diferencias entre modelos de micros.

DIFERENCIAS ENTRE 80387 Y 80287/8087

Según con que modelos de coprocesador queramos que sea compatible nuestro programa , debemos saber que limitaciones impone cada modelo , que serán mayores , naturalmente , contra más viejo sea el modelo de coprocesador.

Las diferencias tanto afectan a la presencia o no de determinadas instrucciones (pues cada nuevo modelo siempre aporta algunas de nuevas) como al funcionamiento de instrucciones ya exitentes en modelos anteriores , así por ejemplo , mientras que FXTRACT produce una excepción de división por 0 en el 387 cuando se le da un operando con valor 0 , en el 287 no se produce error.

A continuación se describen las instrucciones del 387 que no están en el 80287/8087: FPREM1 , FUCOM/FUCOMP/FUCOMPP , FSIN , FCOS y FSINCOS. Como se puede observar , las tres últimas es muy importantes tener presente que no existen en dichos modelos.

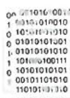
Diferencias de ejecución:

1-FPREM no daba flags de condición de salto fiables en los 87/287 mientras que si lo hace en el 387.

2-FBSTP , FDIV , FIST , FISTP y FSQRT pueden operar en el 387 con operandos no normalizados , lo cual no era posible en los anteriores modelos.

3-FPTAN pone el bit 10 de la CW (palabra de control) a 1 cuando se indica una una operación incompleta solo en el 387.

4-FLD no genera error de no-normalizado en los 87/287 , en cambio en un 387 se produce dicho error al cargar un no-normalizado.



5-FSCALE , FPTAN , FPATAN y F2XM1 tienen rangos de cálculos más amplios.

6-La instrucción FNOP del 387 equivale a la instrucción FSETPM del 287.

7-En un 387 , cuando se carga un no normalizado , se convierte a precisión extendida , en cambio , en el 287 se convierte a no normalizado.

El 387 aporta como novedad funciones trigonométricas

Diferencias generales:

1-En el 387 , cuando se realiza un FINIT , los bits IE y ES de la SW (palabra de estado) se ponen a 1 y el bit IM de la CW se pone a 0 (como indicador de 387 presente).

2-El 80287 sólo tiene el tipo de NaN de señalización, el 387 soporta también el tipo NaN silencioso.

3-Cuando realiza una operación no válida , FSQRT , FDIV y FPREM no producen una excepción en el 387 , en cambio si lo hace un 287.

4-En el 387 se soporta sólo la interpretación afín del infinito , el 287/87 soporta además el modo proyectivo.

5-Cuando se produce una excepción de pila en el 387 , se pone a 1 el bit 6 y 10 de la SW (palabra de estado) y se produce la excepción correspondiente , en cambio , en el 87/287 sólo se produce la excepción.

6-En el 387 , los flags de condición de la SW se ponen a 0 en tres casos: Al ejecutar FINIT , al ejecutar una FPREM incompleta y cuando realizamos una reset hardware. En el 287 solo se reinicializan por hardware.

7-El 387 genera siempre excepción cuando encuentra uno de los tipos de datos que no soporta: pseudo Nan , pseudo-cero , pseudo-infinito y no normalizado. En cambio , el 80287 soporta estos tipos y no da excepciones asociadas.

Diferencias especificas entre 387 y 87:

-Las instrucciones del 8087 FENI/FNENI , FDISI/FNDISI son ignoradas por el 387 (no se ejecutan).

-El modo de procesar las excepciones son muy diferentes en el 8087 respecto al 387 , mientras que el 87 puede procesarlas mediante un vector de interrupción , en el 387 se debe realizar a través de un vector de interrupción 16.

-Los punteros de excepción almacenados en FIP , FCS ,FOO y FOS son diferentes en un 87 que los almacenados en un 387.

-Sólo se pueden generar fallos de segmentación cuando una instrucción o un operando suyo está fuera de límite de segmento en un 387.

-Una instrucción de coprocesador sólo se puede crear una excepción 7 de emulación o tarea conmutada en un 387. Por cierto , esto se realiza mediante los bits EM y TS del registro CR0 del 386.

-En el 387 , cuando se llama a una rutina de tratamiento de excepción , CS:IP contienen el puntero al inicio de la instrucción incluyendo sus prefijos , lo que no pasa en el 87.

-En el 386/387 , la ejecución de las instrucciones se sincroniza automáticamente , por lo tanto , la instrucción *wait* no es necesaria. En cambio en el 8087 , se debía colocar tras una operación de coprocesador para asegurar que el 8086 recibía el resultado.

RESUMEN

En este artículo se ha explicado la evolución de los coprocesadores x87 de Intel, se ha detallado como funcionan internamente y los registros que forman el chip. También se han listado los tipos de datos que soporta así como la historia de como apareció el estándar IEEE de punto flotante. Después se ha listado todo el set de instrucciones que posee el 387 así como la utilidad de cada una de ellas para tener una referencia de las posibilidades del chip.

Una vez aclarados todos los conceptos base , se han dado unos cuantos ejemplos para la comprensión del contenido y unas notas sobre los errores más comunes que se pueden producir en la programación del coprocesador.

Para finalizar , se han detallado todas las diferencias que hay entre el 80387 y el resto de sus antecesores , que son el 8087 y el 80287 , lo cual es bastante importante para evitar errores inesperados. Próximamente Sólo Programadores tiene la intención de continuar con este tipo de artículos no englobados dentro de ningún de los cursos habituales de la revista y especialmente orientados a temas muy a bajo nivel. En el mes de Julio se estudiará algoritmos de compresión y más concretamente el algoritmo de Huffman.

OCULTACIÓN DE CARAS EN 3D

Pedro Antón Alonso

En este número se continúa con las 3D. En esta ocasión se explicará cómo ocultar las caras de un objeto tridimensional.

Hace ya un par de números que se comenzó con las 3D, y quizás el lector se pregunte, qué ha ocurrido con aquella introducción. Bien, se intentaran alternar los artículos de 3D, con otros de igual o mayor interés, para no aburrir a aquellos seguidores de este curso que no están muy interesados en la tres dimensiones. En la *demoScene*, las tres dimensiones tienen tantos seguidores como detractores. Por este motivo, se alternarán los artículos.

LA TERCERA DIMENSIÓN

Como ya se vio en la anterior entrega de tres dimensiones, los puntos con los que se trata tienen tres coordenadas, y la pantalla del ordenador dos. Por lo tanto, algo que cualquiera puede deducir, es la posibilidad de que existan superficies u objetos superpuestos. En este artículo se tratará la forma más utilizada, quizás por ser la más rápida, de representar objetos en tres dimensiones. Cuando se habla de un objeto, por supuesto, no se habla de una serie de puntos que conforman una imagen. En el artículo anterior, la esfera venía definida por puntos, pero, que duda cabe, que ese tipo de representación no deja de ser algo irreal. Nadie ha visto un objeto físico formado únicamente por puntos, ¿no?. Bromas aparte, es evidente que esos puntos han de ir conexionados de alguna forma, y que esa forma no es aleatoria. La correcta interconexión entre los puntos que representan la esfera, conforman la figura sólida, que es lo que en realidad se desea representar.

DEFINICIÓN DE UN OBJETO

Una vez explicado esto, puede verse cómo un objeto queda ahora definido por un número determinado de puntos unidos entre sí, de la forma adecuada. Estas uniones conformarán las caras del objeto.

Pero, ahora surge un nuevo problema. Un objeto puede venir definido por caras con diferentes números de puntos, es decir, las caras de un cubo son cuadrados, pero un icosaedro tendrá caras triangulares. Visto así, bastaría con definir todas las caras, según el objeto que se desee representar, pero si el objeto no está formado por superficies de igual número de lados, se debe llegar a algún tipo de acuerdo en la definición de las caras.

Como la unidad mínima de superficie es el triángulo, no se debe olvidar que un plano se puede determinar, a partir de tres puntos, por lo tanto se creará cualquier superficie a partir de triángulos.

A continuación se verán algunos métodos para ordenar los planos que definen un objeto tridimensional.

MÉTODO DE PREORDENACIÓN

El método de preordenación es especialmente útil si el objeto no se va rotar, o si lo hace alrededor del eje Z.

Consiste en preordenar las caras según se vayan a ver o no, e incluso, eliminar aquellas que nunca se vayan a ver. Esto es, no definir en la matriz, aquellos puntos que vayan a estar en la parte totalmente opuesta al ángulo de visión. De esta forma, el objeto queda muy simplificado. El problema que tiene este método es la limitación de movimiento espacial. Si se eliminan aquellas caras que nunca se van a ver, se debe estar seguro de ello, es decir, producir un giro de 180 grados alrededor del eje Y tendrá como consecuencia una incorrecta visión del objeto. Esto puede solucionarse generando distintas matrices para cada objeto, dependiendo del ángulo de visión. Pero la cantidad de matrices a crear puede ser realmente importante.

Este método, no es muy empleado, salvo en casos muy puntuales, donde el movimiento del objeto está muy limitado. La demo ganadora del Assembly 95, según la opinión del autor, usa este método para representar la abeja del principio de su demo.

MÉTODO DE LA NORMAL

El método de la normal se basa en el producto vectorial de dos vectores.

Para explicar este efecto, primero es necesario recordar la definición de producto vectorial: "El producto vectorial de dos vectores, es otro vector, de dirección perpendicular al plano formado por estos, sentido el de la regla del sacacorchos, y módulo, el producto de sus módulos por el seno del ángulo que forman".

Esta definición va a ser muy importante a partir de ahora, en el tratamiento de las tres dimensiones. El módulo de este vector no es importante, pero la dirección y, sobre todo, el sentido, es lo que indicará si el plano es visible para el observador o no.

Y surge un nuevo problema, si el sentido del vector viene dado por la regla del sacacorchos. Es evidente que el producto vectorial no será conmutativo, es decir, no es lo mismo multiplicar el vector *a* por el vector *b*, que el vector *b* por el vector *a*. Por lo tanto, la definición de las caras del objeto, deberá acatar esta propiedad de no-conmutatividad, debiendo almacenar el orden de las caras de forma correcta. De no hacer esto, nunca se sabrá con certeza si la cara es visible o no. La solución del producto vectorial puede consultarse en el listado 1.

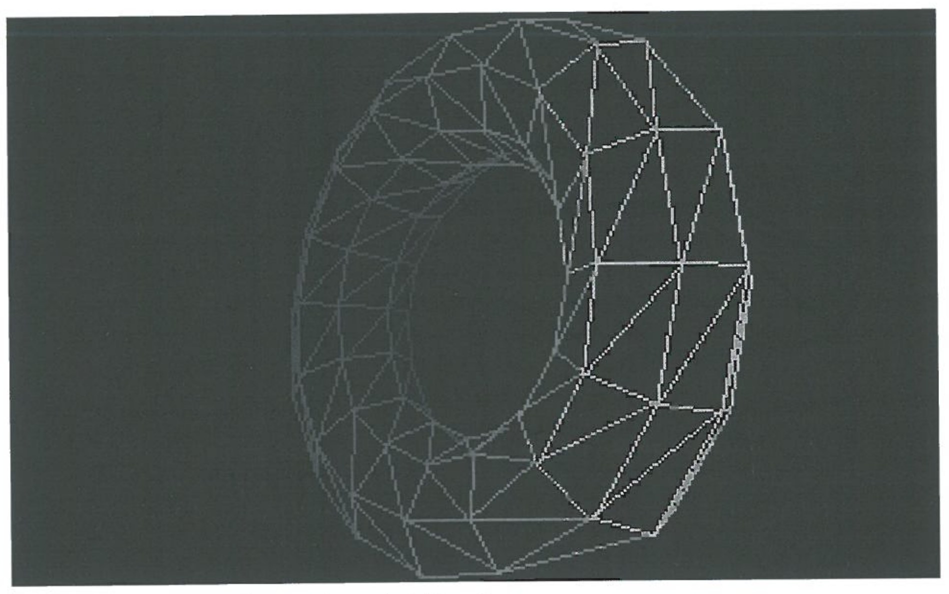
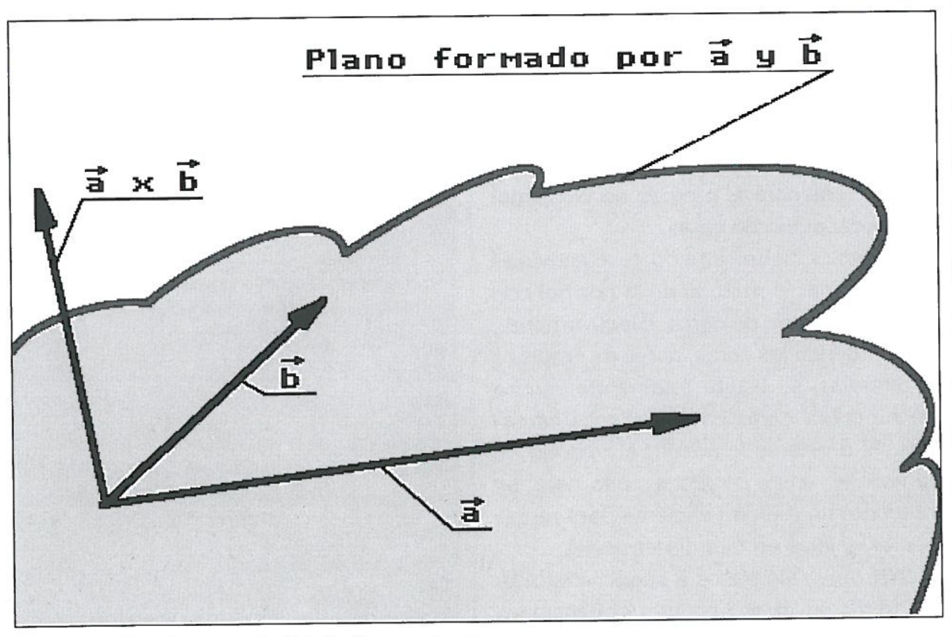


Figura 1.- Ejemplo de figura a base de triángulos.

Solucionado este problema, se debe recordar cómo obtener los vectores sabiendo las coordenadas de los puntos. Recordando las matemáticas elementales, las coordenadas de los vectores, así como el resultado matemático del producto vectorial, vendrán dadas por las operaciones indicadas en la figura 3.

Como se puede observar en dicha figura, es sencillo calcular el producto vectorial de dos vectores, pero resultará aún más sencillo, ya que en realidad, sólo interesará la coordenada z de dicho producto, que es la que indicará, con su signo, si el plano será visible o no, ya que en el caso de que la cara tenga una normal de componente z negativa, significará que está orientada en dirección opuesta a la visión del observador. La implementación de este método puede consultarse en el listado 2. Además, como el lector puede observar, dicha implementación resulta sencilla y ahorrará el tiempo empleado en pintar superficies no visibles.

El inconveniente de este método puede surgir si las superficies son convexas, en cuyo caso, puede ocurrir que un plano visible que se encuentre detrás de otro, a su vez también visible, se pinte encima (figura 4).



Listado 1: Producto vectorial de dos vectores.

MÉTODO DE LA ORDENACIÓN DE PLANOS

La ordenación de planos, junto con el método de la normal, será el método empleado en este curso.

Ordenar los planos en profundidad, junto al método de la normal, será la solución óptima, ya que, de esta forma, se soluciona el problema que tiene el método de la normal.

Si se ordenan los planos en orden decreciente en Z, es decir, los planos más alejados del observador primero, y los más cercanos al final, se logrará que nunca un plano que se encuentre a mayor distancia del observador tape a otro que está más cerca. Esto es evi-

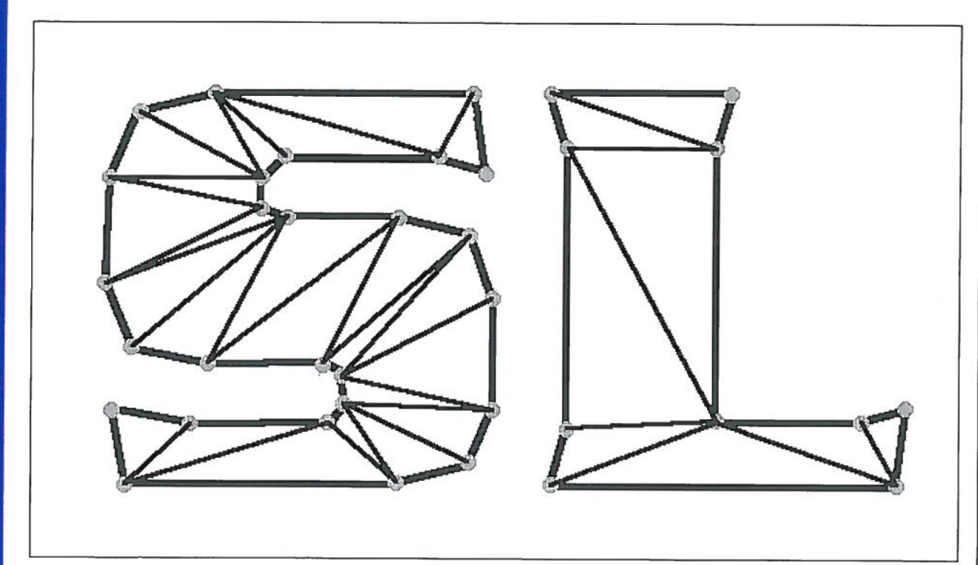


Figura 2.- Triangulado de una superficie cualquiera.

dente ya que, a la hora de pintar las caras, se pintarán primero las que se encuentren más alejadas del observador, ocultando a éstas, si es el caso, las más próximas a éste.

Pero para ello debe conocerse cual es la coordenada Z de cada cara, es por ello que el tipo *Face3D* añadido en la librería, es un array de cuatro enteros. Son cuatro porque en el cuarto de ellos se calculará la coordenada Z media de la cara. Esto se hará hallando la media aritmética de las coordenadas Z de los puntos que definen el plano.

FUNCIÓN VISIBLE

```
Function Visible (X1,Y1,X2,Y2,X3,Y3:Integer)
:Boolean;assembler;
Asm
  mov cx,X2 { reg dx = (X2-X1)*(Y3-Y1)) }
  mov bx,X1
  sub cx,bx
  mov ax,Y3
  mov bx,Y1
  sub ax,bx
  imul cx
  mov dx,ax
  push dx
  mov cx,X3 { reg ax = (X3-X1)*(Y2-Y1) }
  mov bx,X1
  sub cx,bx
  mov ax,Y2
  mov bx,Y1
  sub ax,bx
  imul cx
  pop dx
  sub dx,ax {(X2-X1)*(Y3-Y1)-(X3-X1)*(Y2-Y1)}
  js @@InVisible { Si negativo = Invisible,}
  mov ax,01
  jmp @@Fin
@@InVisible:
  xor ax,ax
@@Fin:
End;
```

Listado 2

En este método es muy importante que el algoritmo a emplear para ordenar las caras sea lo suficientemente rápido. Por ello, se empleará el *QuickSort* (listado 3), que es el algoritmo más rápido cuando el número de caras es elevado, y puesto que los objetos a tratar no serán simples, este algoritmo descarta al *BubleSort,* que sería el otro candidato(algoritmo que, por otra parte, se torna más eficiente cuando el número de elementos a ordenar es pequeño). Se incluirá en la librería, entonces, un algoritmo de ordenación de caras específico a dicho propósito.

El inconveniente de este método es la increíble cantidad de tiempo que se pierde dibujando caras que, posteriormente, van a ser cubiertas por otras más cercanas al observador, ya que con este método, se pintan tanto las caras que son visibles como las que no lo son.

MÉTODO FINAL

Se combinará el método de la normal y la ordenación de caras.

El lector habrá intuido la efectividad de combinar el método de la normal con el la ordenación de caras. Efectivamente, si se ordenan las caras, antes de empezar a pintarlas, se estará totalmente seguro de no cubrir caras con otras más cercanas. Si a esto se le añade, el método de la normal, antes de pintar cada cara, se obtendrá un método eficiente para representar objetos en tres dimensiones.

Así pues, los pasos a seguir a la hora de tratar un objeto en tres dimensiones serán:

- Calcular la rotación de los puntos.
- Proyectar y trasladar dichos puntos.
- Ordenar en Z las caras del objeto.
- Dibujar cada cara del objeto, si su vector normal tiene una componente z positiva.

El código correspondiente a este tratamiento del objeto puede consultarse en el listado 4.

OPTIMIZACIONES Y AMPLIACIONES DE LA LIBRERÍA

La librería de 3D ha sufrido unos cuantos cambios. Se han modificado unas cuantas líneas de código en la

QUICK SORT

```
Procedure QuickSortZ
      (NumPoints:Word;
       Points2qS:Array of Point3D;
    Var Faces2qS :Array of Face3D);
Var
  Cnt       : Word;
{ Specific sort }
  Procedure QSort (First, Last: Integer;
            Var Split1: Integer;
            Var Split2: Integer);
  Var
    AuxFace   : Face3D;
    ZVal      : Integer;
  Begin
    ZVal := Faces2qS[(First+Last) shr 1,4];
    Repeat
      While Faces2qS[First,4] < ZVal do Inc (First);
        While Faces2qS[Last,4] > ZVal do Dec (Last );
          If First <= Last then
            Begin
              AuxFace        := Faces2qS[First];
              Faces2qS[First] := Faces2qS[Last];
              Faces2qS[Last]  := AuxFace;
              Inc (First);
              Dec (Last);
            End;
    Until First > Last;
    Split1 := First;
    Split2 := Last;
  End;
{ Generic QuickSort }
  Procedure QuickSort (First, Last:Integer);
  Var
    SplitPt1, SplitPt2: Integer;
  Begin
    If First < Last then
      Begin
      qSort (First, Last, SplitPt1, SplitPt2);
      If SplitPt1 < Last then QuickSort (SplitPt1, Last);
      If First < SplitPt2 then QuickSort (First, SplitPt2);
      End;
end;
{QuickSortZ}
Begin
 { Calculate CenterZ coor. for each face }
 For Cnt :=0 to NumPoints do
 Faces2qS [Cnt,4]:= (Points2qS[Faces2qS[Cnt,1],3]+
                  Points2qS[Faces2qS[Cnt,2],3]+
                  Points2qS[Faces2qS[Cnt,3],3]) div 3;
 { reOrder it QUICKLY!!!! }
  QuickSort (0,NumPoints);
End;
```

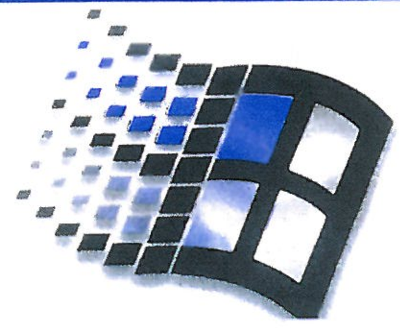
Listado 3

LOS LIBROS DE INFORMÁTICA MÁS ACTUALES AL MEJOR PRECIO

WINDOWS 95











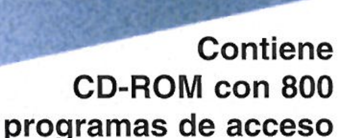
EXPLORANDO INTERNET

las Autopistas de la Información





Contiene CD-ROM con una selección de ficheros de sonido, vídeo e imágenes, así como las utilidades disponibles actualmente para Windows 95.

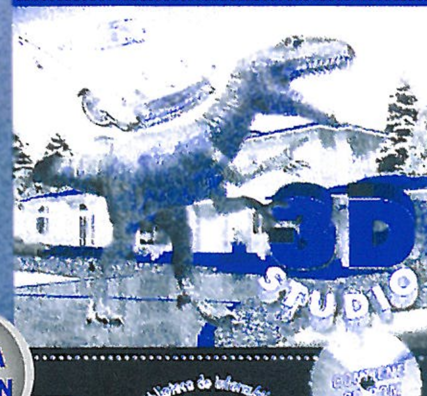


Contiene CD-ROM con 800 programas de acceso rápido a INTERNET



3D STUDIO V.4

Las tres dimensiones a su alcance



CADA LIBRO POR SÓLO:

1.995 ptas.
IVA INCLUIDO



A LA VENTA EN QUIOSCOS, TIENDAS ESPECIALIZADAS O LLAMANDO AL (91) 661 42 11

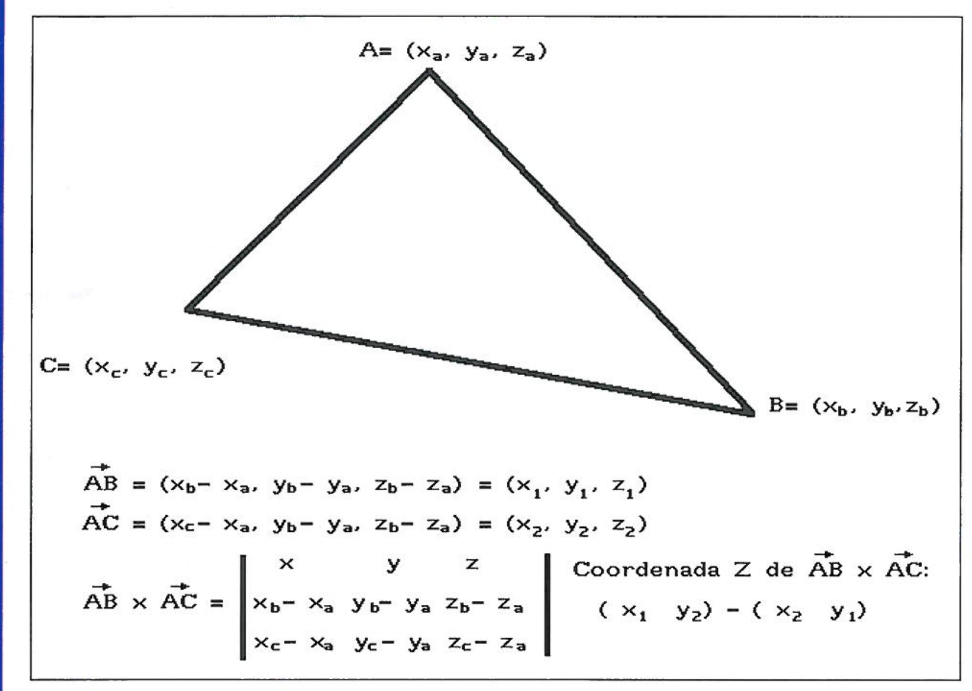


Figura 3.- Componente Z del producto vectorial de 2 vectores.

TRATAMIENTO DE UN OBJETO EN 3D

```
Calc3DRotations (FastSin(XAngle),FastCos(XAngle),
                 FastSin(YAngle),FastCos(YAngle),
                 FastSin(ZAngle),FastCos(ZAngle),
                 Points,Temp,High(Points));
Proyect  (160,100,High(Points),Temp,Temp);
QuickSortZ (High(Faces),Temp,Faces);
For Cnt:=0 to High (Faces) do
  If Visible (Temp [Faces[Cnt,1],1],Temp [Faces[Cnt,1],2],
            Temp [Faces[Cnt,2],1],Temp [Faces[Cnt,2],2],
            Temp [Faces[Cnt,3],1],Temp [Faces[Cnt,3],2])
  then
    Begin
      DrawLine (SegBuffer,
              Temp [Faces[Cnt,1],1],Temp [Faces[Cnt,1],2],
              Temp [Faces[Cnt,2],1],Temp [Faces[Cnt,2],2],
              Col);
      DrawLine (SegBuffer,
              Temp [Faces[Cnt,1],1],Temp[Faces[Cnt,1],2],
              Temp [Faces[Cnt,3],1],Temp [Faces[Cnt,3],2],
              Col);
      DrawLine (SegBuffer,
              Temp [Faces[Cnt,2],1],Temp [Faces[Cnt,2],2],
              Temp [Faces[Cnt,3],1],Temp [Faces[Cnt,3],2],
              Col);
    End;
```

Listado 4

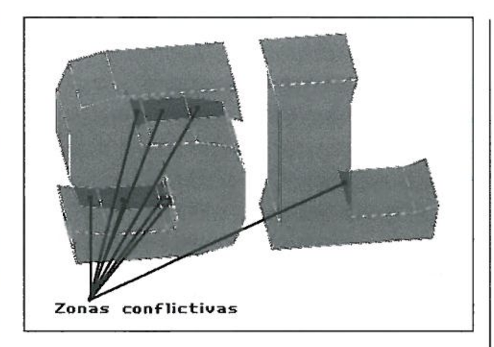


Figura 4.- Problemas del método de la normal.

librería creada en el anterior artículo de 3D.

En primer lugar, se han eliminado las divisiones en el procedimiento *Calc3DRotations*, sustituyendo las divisiones entre 256 por la instrucción *SAR Reg,8*, con la que se obtendrá un resultado idéntico, ahorrando un gran número de ciclos.

Además, se ha modificado el procedimiento *Proyect*, para que éste calcule la proyección de todos los puntos, en lugar de hacerlo de uno en uno.

Como es de suponer, se han añadido procedimientos para gestionar el método de ordenación de planos, y el de la normal. El nombre que toman dichos procedimientos en la librería es *QuickSortZ* y *Visible*. Su uso es muy sencillo, y se ilustra la forma de trabajar con ambos procedimientos en el ejemplo que acompaña a este artículo.

EL EJEMPLO

Este mes se moverá un toro de alambre.

En muchas demos se ven toros moviéndose. Por ahora, habrá que conformarse con un toro de alambre, pero en sucesivos artículos, se irá rellenando dicho toro.

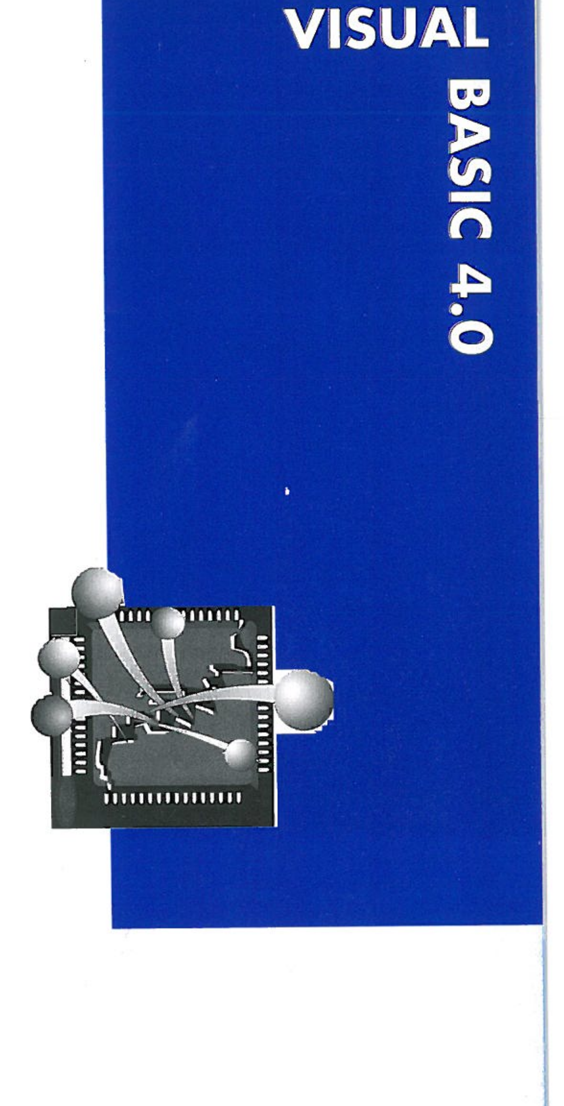
En el código adjunto, se pueden observar los pasos que se han de seguir para mover un objeto sólido en tres dimensiones. El tratamiento a seguir para la elaboración del programa es muy simple:

- a- Crear un degradado en la paleta, para dar mayor sensación de pers pectiva.
- b- Inicializar las variables tridimensio nales de posición del observador en el eje Z (determinará el tamaño del objeto), así como los ángulos de rotación iniciales.
- c- Reservar memoria para el buffer.
- d- Limpiar el buffer, o restaurarlo si se usa un fondo.
- e- Rotar el objeto sobre los tres ángulos.
- f- Proyectar el objeto en dos dimensiones.
- g- Ordenar las caras del objeto en orden de Z decreciente.
- h- Pintar las caras visibles usando el método de la normal.
- i- Esperar el retrazo vertical.
- j- Volcar el buffer a la memoria de vídeo.
- k- Actualizar los ángulos de rotación.
- l- Repetir el proceso desde el paso d, si no se pulsa la tecla de ESC.

Próximamente, se estudiará la forma de rellenar las caras de los objetos usando distintos tipos de rellenado.

EVENTOS Y MÉTODOS

Juan Manuel y Luis Martín



La creación de aplicaciones con Visual Basic se realiza en dos etapas : el diseño gráfico del interfaz con el usuario y la codificación de las instrucciones de la aplicación. En los artículos anteriores se han analizado las técnicas para realizar la primera. El presente artículo describe cómo realizar la segunda

Como se ha venido mencionando a lo largo de los anteriores artículos, la creación de aplicaciones para Windows con Visual Basic 4.0 comprende dos grandes etapas. En primer lugar debe realizarse el diseño de la interfaz de la aplicación con el usuario. Para ello, deben añadirse al proyecto los formularios que intervendrán en la aplicación y situar en ellos los controles necesarios. Ambos tipos de objetos deben ser personalizados mediante la especificación de sus propiedades.

Los objetos, ya sean formularios o controles, disponen de propiedades que permiten definir tanto su aspecto como su comportamiento. Los objetos también incorporan una serie de procedimientos propios, denominados *métodos*, que permiten realizar diversas acciones con ellos. Así, por ejemplo, existen métodos para mover un objeto, para visualizarlo, para pasarle el foco, etc.

Además, dado que Visual Basic es un lenguaje guiado por eventos, los objetos llevarán asociados también otros procedimientos, denominados *procedimientos de evento*. Estos procedimientos deben ser definidos por el programador con el fin de que el objeto responda a un determinado evento.

LOS EVENTOS

Para que una aplicación realizada con Visual Basic sea capaz de realizar distintas acciones no basta con especificar las propiedades de sus objetos. Además, es necesario añadir un cierto código de programa que lleve a cabo dichas acciones. El mecanismo para ejecutar este código son los eventos.

Durante la ejecución de una aplicación se producen numerosos eventos, que pueden ser ocasionados por el usuario, por el sistema o por otras aplicaciones. Para poder dar respuesta a los distintos eventos que se produzcan, los objetos de Visual Basic incorporan unos procedimientos especiales denominados *Procedimientos de evento*. Estos procedimientos deben ser escritos por el programador en lenguaje Basic, y deben codificar las acciones a realizar como respuesta a dicho evento.

Cada objeto de una aplicación creada con Visual Basic es capaz de reconocer un cierto repertorio de eventos. Inicialmente, los procedimientos de evento no contienen ningún código, por lo que no se producirá ninguna acción. El programador debe seleccionar aquellos eventos a los que debe responder cada objeto y codificar las acciones a realizar en sus procedimientos de evento correspondientes.

ESCRITURA DE PROCEDIMIENTOS DE EVENTO

La escritura de los procedimientos de evento de los objetos de una aplicación se realiza en la Ventana de código. Por lo tanto, la primera acción a realizar es acceder a dicha ventana. Esta operación se explicó en el primer artículo. De cualquier forma, el acceso a la misma puede realizarse de múltiples formas:

- Realizando un doble clic sobre el formulario o sobre cualquier control del mismo.
- Seleccionando la opción *Código* del menú *Ver*.

• Haciendo clic sobre el botón *Ver Código* en la Ventana de proyecto.
• Pulsando la tecla [F7].

Cuando se realiza alguna de las acciones enumeradas anteriormente se despliega un cuadro de diálogo como el mostrado en la figura 1. Dicho cuadro de diálogo está compuesto por tres partes:

• La *Lista de objetos*: Contiene una lista con los nombres de todos los objetos incluidos en el formulario activo, incluido el propio formulario.
• La *Lista de procedimientos*: Contiene una lista con todos los eventos del objeto seleccionado en la Lista de objetos. Si alguno de los procedimientos de evento de dicho control ya se ha introducido, el nombre del evento será visualizado en negrita para así avisar al programador de dicha circunstáncia.
• El *Área de edición*: Es el lugar donde se realiza la edición del procedimiento de evento actualmente seleccionado.

En un principio, la ventana mostrará el procedimiento de evento por defecto para el control actualmente seleccionado. Si exite ya un procedimiento de evento para ese control que no se encuentre vacío, será éste el que se visualizará. Cuando se selecciona un objeto y un evento de las correspondientes listas, en el área de edición se muestra una plantilla del correspondiente procedimiento de evento. Dicha plantilla consta de dos líneas de código que marcan el comienzo y el final del procedimiento de evento. Se trata de las sentencias *Sub* y *End Sub*. Entre ellas deberán situarse las instrucciones encargadas de describir todas aquellas aciones a realizar como respuesta al evento que se ha producido. Así, la sintaxis general de los procedimientos de evento de cualquier objeto que forme parte de la aplicación puede resumirse de la siguiente forma:

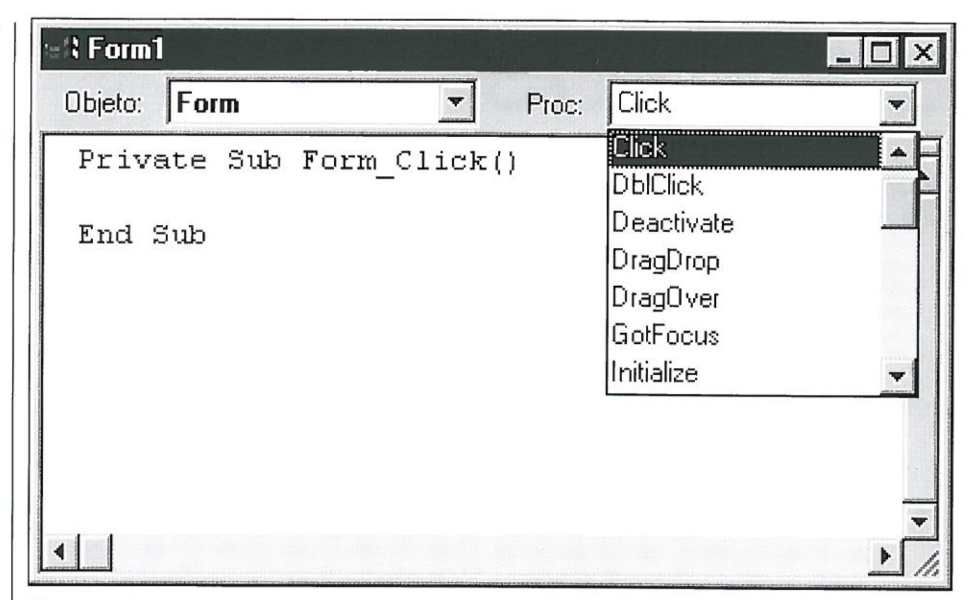


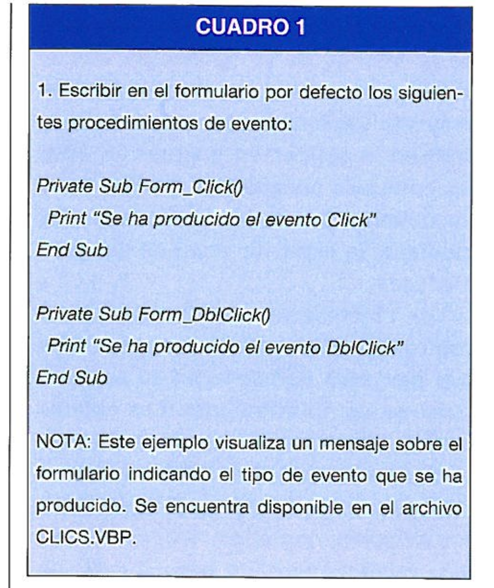
Figura 1. Aspecto de la Ventana de edición.

Private Sub
NombreObjeto_NombreEvento
([Argumentos])
[Definición de variable locales]
...
[Instrucciones]
...
End Sub

las palabras reservadas *Private* y *Sub* indican el tipo de procedimiento de que se trata. El programador debe tener presente que los procedimientos de evento son siempre subrutinas privadas, de ahí la inclusión automática de dichas palabras. Tras ellas, aparece el nombre del procedimiento de evento que se compondrá de dos partes: el nombre del objeto y el nombre del evento, separados por el carácter subrayado (_). Así, por ejemplo, el procedimiento de evento del cuadro de texto *Text1* que responde al evento *Change* se llamará *Text1_Change*.

El mecanismo para ejecutar código en Visual Basic son los procedimientos de evento

Algunos procedimientos de evento también pueden incluir argumentos con información adicional al evento producido. Además, también es posible definir variables locales dentro de ellos para su posterior utilización en el código.



CUADRO 1

1. Escribir en el formulario por defecto los siguientes procedimientos de evento:

Private Sub Form_Click()
Print "Se ha producido el evento Click"
End Sub

Private Sub Form_DblClick()
Print "Se ha producido el evento DblClick"
End Sub

NOTA: Este ejemplo visualiza un mensaje sobre el formulario indicando el tipo de evento que se ha producido. Se encuentra disponible en el archivo CLICS.VBP.

Ejemplo de los eventos Click y DblClick.

Como se ha mencionado anteriormente, cada uno de los objetos de una aplicación incorpora su propio repertorio de procedimientos de evento. Debido a ello, los eventos que puede detectar cada uno de los objetos son, en general, distintos. Sin embargo, existen una serie de eventos que son comunes a la mayoría de los controles y formularios. Algunos de ellos están relacionados con temas específicos, por lo que serán analizados en posteriores artículos. Dentro de ellos pueden inclñuirse los procedimientos de evento relacionados con las operaciones de arrastre y soltado de objetos, la pulsación de teclas, el enlace con otras aplicaciones, etc.

LOS EVENTOS CLICK Y DBLCLICK

Estos eventos son comunes tanto a controles como a formularios, y se producen cuando el usuario realiza una pulsación de los botones del ratón.

El evento *Click* ocurre cuando se pulsa el botón izquierdo del ratón sobre un objeto. En el caso de los formularios, la pulsación deberá realizarse sobre una zona del área de cliente en la que no haya ningún control, o en la que exista un control deshabilitado (con su propiedad *Enabled* a *False*).

Además, este evento puede producirse por cualquiera de las siguientes causas, siendo tratado de la misma forma:

Inicialmente, dado que los procedimientos de evento están vacíos no se producirá ninguna acción

- Cuando se pulsa el botón derecho del ratón, excepto en los botones de comando, casillas de verificación y botones de opción.
- Cuando se selecciona un elemento de un control tipo lista, ya sea con el ratón o con el teclado.
- Cuando se pulsa [ESPACIO] estando el foco en un botón de comando, una casilla de verificación o un botón de opción.
- Cuando se pulsa [INTRO] en un formulario con un botón de comando que tiene su propiedad *Default* a *True*.
- Cuando se pulsa [ESCAPE] en un formulario con un botón de comando que tiene su propiedad *Cancel* a *True*.
- Cuando se pulsa la tecla de acceso de un control.
- Cuando se pone a *True* la propiedad *Value* de un botón de comando, una casilla de verificación o un botón de opción mediante código.

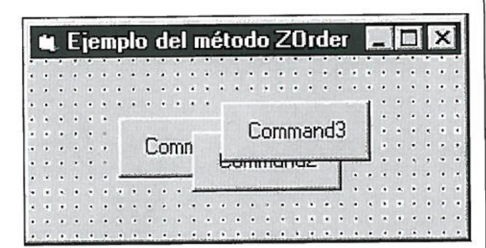


Figura 2.- Solapamiento de controles sobre el formulario.

El evento *DblClick* se produce únicamente cuando el usuario realiza un doble clic sobre un determinado objeto con el botón izquierdo del ratón. En el caso de los formularios, el doble clic también debe realizarse en una zona del mismo vacía de controles. El cuadro 1 muestra un ejemplo de la utilización de ambos eventos.

LOS EVENTOS GOTFOCUS Y LOSTFOCUS

Las aplicaciones Windows sólo una ventana se encuentra activa en cada momento. Dentro de ella, sólo uno de sus controles será el que tenga el foco. El control que tiene el foco en un determinado momento es el único capaz de interactuar con el usuario y, generalmente, se visualiza resaltado.

Cuando un objeto recibe el foco se produce el evento *GotFocus*. Igualmente, cuando lo pierde se produce el evento *LostFocus*. Su utilización más habitual es para proporcionar información sobre el objeto al usuario y para comprobar sus datos, respectivamente. El cuadro 2 muestra un ejemplo del funcionamiento de estos eventos.

Estos dos eventos son comunes a controles y formularios. Sin embargo, en el caso de los formularios, el foco sólo podrá ser capturado cuando no contenga ningún control que pueda capturarlo, o cuando todos sus controles se encuentren deshabilitados.

LOS EVENTOS LOAD Y UNLOAD

Para utilizar un formulario en tiempo de ejecución, es necesario realizar una operación previa, mediante la cual el formulario es cargardo en la memoria. Los formularios se cargan en memoria realizando cualquiera de las siguientes acciones:

CUADRO 2

1. Añadir al formulario por defecto dos etiquetas y dos botones de comando.

2. Escribir los siguientes procedimientos de evento:

```
Private Sub Command1_GotFocus()

  Label1.Caption = "El botón Command1 ha capturado el foco"
End Sub

Private Sub Command2_GotFocus()

  Label1.Caption = "El botón Command2 ha capturado el foco"
End Sub

Private Sub Command1_LostFocus()

  Label2.Caption = "El botón Command1 ha perdido el foco"
End Sub

Private Sub Command2_LostFocus()

  Label2.Caption = "El botón Command2 ha perdido el foco"
End Sub
```

NOTA: Este ejemplo muestra mensajes indicando el botón que ha capturado el foco y el que lo ha perdido. Se encuentra disponible en el archivo FOCO.VBP.

Ejemplo de los eventos GotFocus y LostFocus.

- Cuando se trata del formulario de arranque de la aplicación.
- Cuando se utiliza su método *Show* en el código.
- Cuando se utiliza la sentencia *Load*.
- Cuando se hace referencia a alguna propiedad de éste o de sus controles, sin estar aún cargado.

De la misma forma, los formularios de descargarse de la memoria. Esta acción puede realizarse mediante cualquiera de los siguientes formas, produciendo todas ellas el mismo efecto:

- Cuando el usuario lo cierra (con el botón cerrar o el menú de control)
- Cuando se utiliza la sentencia *Unload*.

CUADRO 3

1. Escribir en el formulario por defecto los siguientes procedimientos de evento:

```
Private Sub Form_Load()
 Form1.Left = (Screen.Width - Form1.Width) / 2
 Form1.Top = (Screen.Height - Form1.Height) / 2
End Sub

Private Sub Form_Unload(Cancel As Integer)
 If MsgBox("¿Desea Salir ?", vbOkCancel, "SALIDA") = vbCancel Then
  Cancel = True
 End If
End Sub
```

NOTA: Este ejemplo utiliza el evento Load para centrar el formulario en la pantalla, y Unload para pedir confirmación de la salida. Se encuentra disponible en el archivo CARGA.VBP.

Ejemplo de los eventos Load y Unload.

CUADRO 4

C1. Añadir un cuadro de texto al formulario por defecto.
2. Especificar las siguientes propiedades:

```
Text1.MultiLine = True
Text1.ScrollBars = 3 (Ambas)
```

3. Escribir el siguiente procedimiento de evento:

```
Private Sub Form_Resize()
 Text1.Left = 0
 Text1.Top = 0
 Text1.Width = Form1.ScaleWidth
 Text1.Height = Form1.ScaleHeight
End Sub
```

NOTA: Este ejemplo sitúa un cuadro de texto ocupando todo el área de cliente del formulario, independientemente del tamaño de éste. Se encuentra disponible en el archivo RESIZE.VBP

Ejemplo del evento Resize.

CUADRO 5

1. Añadir una imagen (Image) al formulario por defecto.
2. Especificar la siguiente propiedad:

```
Image1.Picture = ICONS\INDUSTRY\PLANE.ICO
```

3. Escribir el siguiente procedimiento de evento:

```
Private Sub Image1_Click()
 Image1.Move Image1.Left + 240, Image1.Top + 240
End Sub
```

NOTA: En este ejemplo el icono de un avión se mueve por la pantalla cuando se hace clic sobre él. Se encuentra disponible en el archivo MOVER.VBP.

Ejemplo del método Move.

- Cuando se cierra la aplicación desde la Barra de tareas.
- Cuando se cierra Windows.

Una vez cargado un formulario en la memoria se produce el evento *Load*. Este evento suele utilizarse para operaciones de inicialización de variables y controles, carga de listas, etc.

La escritura de los procedimientos de evento de una aplicación se realiza en la Ventana de código

Igualmente, antes de descargarse el formulario se produce el evento *Unload*. A diferencia del anterior, su procedimiento de evento dispone del argumento *Cancel*, que permite abortar la operación de descarga. Para ello, bastará con asignarle un valor distinto de cero. El cuadro 3 muestra un ejemplo de utilización de ambos eventos.

LOS EVENTOS ACTIVATE Y DEACTIVATE

En tiempo de ejecución, sólo uno de los formularios de la aplicación se encuentra activo, visualizándose resaltada su barra de título. El formulario activo será aquel que contenga el control que posee el foco, incluido el propio formulario.

Los formularios pueden activarse por una acción del usuario (cuando hace un clic sobre él) o a través del código (cuando se utilizan los métodos *Show* y *SetFocus*). En ambos casos se produce el evento *Activate*. Igualmente, al desactivarse un formulario se produce el evento *Deactivate*.

El evento *Activate* no se produce cuando se carga un formulario sin visualizarlo. El evento *Deactivate* tampoco se produce cuando éste se descarga, ni cuando se pasa el foco a otra aplicación. El evento *Activate* se produce antes que el evento *GotFocus*, mientras que *Deavtivate* se produce después de *LostFocus*.

EL EVENTO RESIZE

Cuando el tamaño de un formulario cambia, ya sea por acción del usuario o de la propia aplicación, se produce el evento *Resize*. Este evento también se produce la primera vez que se visualiza el formulario. Generalmente, este evento se utiliza para recalcular los valores de algunas variables y propiedades que dependen del tamaño del formulario. De esta forma, es posible mantener tanto el tamaño como la posición de los controles de forma proporcional al del formulario. El cuadro 4 muestra un ejemplo de la utilización de este evento.

LOS MÉTODOS

Los objetos disponen también de un repertorio de procedimientos propios, llamados métodos, que permiten realizar diversas acciones sobre ellos. Así, existen métodos para mover objetos, dibujar sobre ellos, capturar el foco, etc.

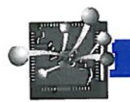
La sintaxis general de un método puede resumirse de la siguiente forma:

Objeto.Método [Argumentos]

Aunque los métodos suelen ser bastante específicos del tipo de objeto al que pertenecen, existen también algunos de ellos comunes a la mayoría de los objetos.

EL MÉTODO MOVE

Para mover de posición un objeto en la pantalla se utiliza su método *Move*. Además, este método también permite alterar el tamaño del objeto de forma similar a como se realiza cuando se alteran los valores de sus propiedades. Su sintaxis es la siguiente:



CUADRO 6

1. Añadir 3 botones de comando y tres cuadros de texto al formulario por defecto.
2. Escribir los siguientes procedimientos de evento:

```
Private Sub Command1_Click()
  Text1.SetFocus
End Sub

Private Sub Command2_Click()
  Text2.SetFocus
End Sub

Private Sub Command3_Click()
  Text3.SetFocus
End Sub
```

NOTA: En este ejemplo, cuando se hace clic sobre uno de los botones de comando, el foco pasa al cuadro de texto correspondiente. Se encuentra disponible en el archivo PASAFOCO.VBP.

Ejemplo del método SetFocus.

CUADRO 7

1. Añadir tres botones de comando al formulario por defecto, tal y como se ve en la figura 2.
2. Escribir los siguientes procedimientos de evento:

```
Private Sub Command1_Click()
  Command1.ZOrder
End Sub

Private Sub Command2_Click()
  Command2.ZOrder
End Sub

Private Sub Command3_Click()
  Command3.ZOrder
End Sub
```

NOTA: Este ejemplo muestra tres botones apilados, de forma que, al hacer clic sobre uno de ellos, éste se sitúa delante de los demás. Se encuentra disponible en el archivo APILAR.VBP.

Ejemplo del método Zorder.

Objeto.Move Izquierda[, Arriba[, Anchura[, Altura]]]

Los argumentos *Izquierda* y *Arriba* indican la nueva posición de la esquina superior izquierda del objeto. Los argumentos *Anchura* y *Altura* indican las nuevas dimensiones del objeto. Sólo el argumento *Izquierda* es obligatorio. Así, por ejemplo, para especificar sólo una nueva anchura puede utilizarse la siguiente instrucción:

Command1.Move Command1.Left, , 300

El cuadro 5 muestra un ejemplo de la utilización de este método.

EL MÉTODO SETFOCUS

Para hacer que un objeto capture el foco se utiliza su método *SetFocus*. Este método permite alterar el orden preestablecido en que los objetos de una aplicación van capturando el foco siguiendo las pulsaciones de las teclas TAB y SHIFT+TAB. Su sintaxis es la siguiente:

Objeto.SetFocus

No todos los tipos de controles pueden capturar el foco. Algunos ejemplos de este tipo de controles pueden ser las etiquetas, las líneas, los marcos, etc. Además, para que un control capture el foco es necesario que éste se encuentre visible y habilitado. El cuadro 6 muestra un ejemplo de utilización de este método.

Los formularios no podrán capturar el foco, salvo en los siguinetes casos:

- Si el formulario no contiene ningún control.
- Si el formulario sólo contiene controles que tampoco puedan capturar el foco.
- Que los controles que puedan obtener el foco se encuentren deshabilitados.

Los eventos GotFocus y LostFocus se producen cuando un objeto obtiene y pierde el foco, respectivamente

EL MÉTODO ZORDER

Al situar los objetos sobre la pantalla, estos pueden quedar solapados unos sobre otros. De esta forma, los objetos que se encuentran delante ocultarán total o parcialmente a los que se encuentran detrás. La figura 2 muestra esta un ejemplo de esta situación.

En tiempo de diseño, es posible cambiar el orden en que los objetos se encuentran apilados utilizando las siguientes opciones del menú edición o las teclas rápidas indicadas entre paréntesis:

- *Traer al frente* ([CTRL+J]) : Sitúa el control seleccionado delante de los demás.
- *Enviar hacia atrás* ([CTRL+K]) : Sitúa el control seleccionado detrás del resto.

Estas dos operaciones también pueden realizarse en tiempo de ejecución, aunque para ello será necesario utilizar el método *Zorder* de cada uno de los objetos. La sintaxis de dicho método es la siguiente:

Objeto.Zorder [Posición]

Cuando se especifica un valor 0 en el argumento *Posición*, el objeto se sitúa delante de los demás. Por el contrario, cuando se especifica un valor 1, se sitúa detrás. El valor por defecto es cero. El cuadro 7 muestra un ejemplo de utilización de este método.

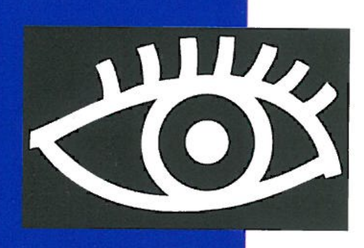
CONCLUSIÓN

El repertorio de métodos y procedimientos de evento de los formularios y los distintos tipos de controles de Visual Basic es realmente amplio. Aquí se han analizado los comunes y más frecuentemente utilizados. Los relacionados con determinados temas, así como los específicos de cada tipo de control serán analizados en posteriores artículos.

NOTA: Los ejemplos incluidos en el disco han sid realizados con la versión de 32 bits de Visual Basic 4.0. El lector no debe preocuparse si posee la versión de 16 bits, pues Visual Basic realiza la sustitución de los controles de 32 bits por los de 16 bits de forma automática.

LA CLASE CWINAPP

Jorge R. Regidor



Sin duda alguna, la clase más importante a la hora de crear aplicaciones para Windows utilizando Visual C++ 4.0 es *CWinApp*, la cual permite crear y manipular la aplicación a gusto del programador, derivando la clase de aplicación a partir de ella

Después de haber introducido el modo de trabajo de Visual C++ y haber creado el esqueleto básico de una aplicación, es hora de entrar un poco más en detalle y echar un vistazo a la clase principal de toda aplicación que use la librería MFC, es decir, la clase *CWinApp*.

Como ya se ha visto con anterioridad, *AppWizard* crea la propia clase de aplicación derivada de *CWinApp*, lo que permite sobrecargar algunas funciones miembro de esta clase para adaptarlas a las necesidades, así como añadir cuantas se estimen oportunas.

Como casi siempre, este mecanismo de introducción de nuevo código, es más fácil de implementar, y pasa por sobrecargar algunas funciones, donde además se indica el modo de llamar a la función de la cual derivan. Todo esto hace que programar para Windows en C++ con las MFCs parezca un sueño, pero a veces y fruto del desconocimiento del curso normal de las aplicaciones para Windows, este sueño puede tornarse en pesadilla. En este artículo se va a relatar cómo modificar e introducir novedades en la aplicación, así como de mostrar algunas de las cosas que no se deben hacer.

LA CLASE CWINAPP

Como ya se ha indicado con anterioridad, es de la clase *CWinApp* de la cual debe derivar la aplicación. Esta clase permite inicializar la aplicación y cada una de sus instancias, así como de correr cada una de ellas.

Una aplicación MFC sólo puede tener un objeto derivado de *CWinApp* y, además, debe construirse de forma global, para estar disponible cuando la función *WinMain* sea llamada por Windows.

La clase *CWinApp* deriva de *CWinThread* (Ver Figura A), que representa cada uno de los diferentes hilos de ejecución dentro de una aplicación multitarea de Windows (sólo en 32 bits). El primer y principal hilo de ejecución (o *thread*) creado es *CWinApp*, pero además de éste pueden crearse varios, cada uno con la posibilidad de atender mensajes o no, en función de si son *threads* de trabajo, es decir, sin atender a mensajes, o *thread* de interfaz, con atención a mensajes. La programación de diversos *threads* todavía se escapa del estado actual del curso, por lo que será tratado a posteriori. Casi como nota al margen, destacar que la clase *CWinThread* deriva de la clase *CCmdTarget*, la cual permite enrutar comandos y mensajes de Windows, y esta clase, a su vez deriva de *CObject*, la clase madre dentro de la arquitectura de clases de la librería MFC.

La figura B ilustra la jerarquía de clases de la librería MFC, donde se observa la derivación de la mayoría de las clases de CObject. Además, puede observarse la primera en la siguiente escala de derivación una diferenciación entre las clases que aceptan mensajes, es decir, derivadas de *CCmdTarget*, como *CWinThread*, *CWnd*, *Cdocument*, etc. y las clases que no pueden recibir mensajes como *CException*, las clases de soporte para el *GDI*, *Cfile*, etc. Además, existe un grupo un tanto especial, ya que no derivan de *CObject*, y son las clases de la parte derecha como *CArchive*, *CTime*, *CMemoryState*, etc.

Una vez pasado este inciso, hay que proseguir con el acercamiento a la clase *CWinApp*. Esta clase tiene varias funciones miembro para su manejo, pero,

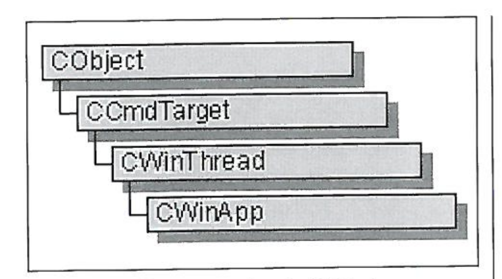


Figura A. Clases madre de CWinApp.

además, existen un grupo de funciones de entorno global que permiten acceder al objeto *CWinApp*. Estas funciones se ilustran en la Tabla 1. Pueden utilizarse desde cualquier punto de la aplicación para obtener información de la misma.

DATOS DE CWINAPP

Los datos internos de la clase *CWinApp*, permiten almacenar y conocer el estado del objeto de aplicación en todo momento. Dichos datos son los siguientes:

• *m_pszAppName*, que permite saber el nombre de la aplicación.

• *m_hInstance*, que guarda el handle de la instancia de la aplicación.

• *m_hPrevInstance*, siempre a NULL en aplicaciones de 32 bits. Antes se utilizaba para saber si existía una instancia de la aplicación ya abierta. Ahora se utiliza la función *CWnd::FindWindow*.

• *m_lpCmdLine*, que contiene la línea de comandos de la aplicación.

• *m_nCmdShow*, que especifica el modo en el cual se debe mostrar la ventana.

• *m_bHelpMode*, indica si el usuario está en modo de ayuda contextual (SHIFT+F1).

• *m_pActiveWnd*, puntero a la ventana principal para la aplicación contenedora cuando está activo el modo de servidor OLE.

• *m_pszExeName*, indica el nombre del modulo (EXE) de la aplicación.

• *m_pszHelpFilePath*, que indica dónde están los ficheros de ayuda.

• *m_pszProfileName*, que indica el nombre del fichero .INI.

• *m_pszRegistryKey*, usado para almacenar los datos de configuración en el registro de Windows.

CONSTRUCTOR DE CWINAPP

La clase *CWinApp* sólo tiene un constructor, el cual tiene el siguiente formato:

CWinApp(LPCTSTR lpszAppName = NULL);

donde *lpszAppName* es el nombre de la aplicación. Si es NULL, *CWinApp* usa la cadena de los recursos *AFX_IDS_APP_TITTLE* o bien el nombre del ejecutable.

Este constructor, almacena un puntero al objeto *CWinApp* que será utilizado en *WinMain* para llamar a las funciones miembro del objeto e inicializar la aplicación.

FUNCIONES MIEMBRO

Dentro de las funciones miembro se pueden diferenciar tres grupos. Aquellas que sirven para realizar operaciones en la aplicación, aquellas que

WinMain
InitApplication
InitInstance
Run
OnIdle
ExitInstance

Figura C. Secuencia de llamadas de la clase CWinApp.

permiten inicializar la misma, y aquellas que pueden ser sobrecargadas y que permiten modificar el funcionamiento de la aplicación.

Las funciones principales que nos permiten realizar operaciones sobre la aplicaciones son:

- *LoadCursor*, *LoadStandardCursor* y *LoadOEMCursor*, que permiten cargar un cursor en la aplicación.

- *LoadIcon*, *LoadStandardIcon* y *LoadOEMIcon*, que permiten cargar el icono de la aplicación.

- *ParseCommandLine*, determina las acciones fruto de las banderas y parámetros de la línea de comandos.

- *ProcessShellCommand*, ejecuta las acciones fruto de los parámetros y las banderas de la línea de comandos.
- *GetProfileInt*, lee un entero de una sección del fichero INI de la aplicación.

- *GetProfileString*, lee una cadena de una sección del fichero INI de la aplicación.

- *WriteProfileInt*, escribe un entero en una sección del fichero INI de la aplicación.

- *WriteProfileString*, escribe una cadena en una sección del fichero INI de la aplicación.

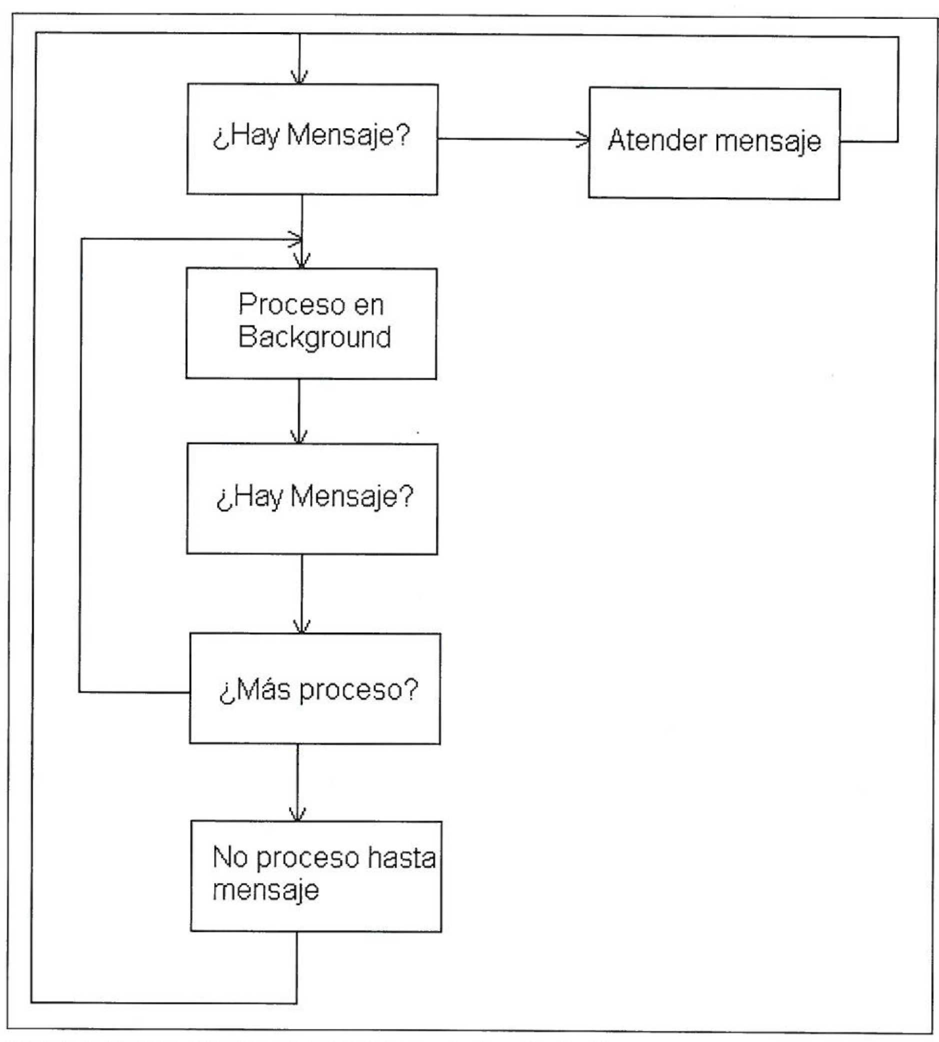


Figura D. Método de proceso de fondo con la función OnIdle.

- *Enable3dControls* y *Enable3dControls Static*, que permiten habilitar los controles con aspecto 3-D cargando la librería dinámica y estáticamente respectivamente.

El objeto derivado de CWinApp debe ser declarado global para poder ser llamado desde WinMain

Dentro del otro grupo de funciones miembro encargadas de la inicialización, pueden destacarse las siguientes:

- *LoadStdProfileSetting*, que carga las opciones por defecto de configuración almacenadas en el fichero INI, así como la lista de los últimos ficheros abiertos.

- *SetDialogBkColor*, que cambia el color por defecto de las cajas de diálogo, permitiendo especificar el color de fondo y de primer plano.

Por último, se va a tratar las funciones virtuales de la clase *CWinApp*, y en ellas se parará un poco más, por ser éstas de mayor interés, dado su capacidad de modifica el comportamiento de la aplicación por defecto.

LA FUNCIÓN MIEMBRO INITAPLICATION

Esta función se utiliza para realizar cualquier tipo de inicialización a nivel de aplicación, es decir, que sólo se ejecutará al cargar por primera vez la aplicación, y no al cargar las instancias o copias de la misma.

LA FUNCIÓN MIEMBRO INITINSTANCE

Esta función es llamada cada vez que se carga una nueva copia de la aplicación. Esta función debe ser sobrecargada, para permitir crear las ventanas propias e inicializar los datos de cada instancia. *AppWizard* ya sobrecarga esta función automáticamente, e introduce el código necesario de inicialización y de creación de las ventanas. Pero es posible modificar o añadir nuevas funcionalidades. En el siguiente ejemplo se ha añadido una llamada a la función *SetDialogBkColor*, estableciendo como color de fondo el amarillo y como color de primer plano rojo. Como se puede apreciar, al ejecutar la aplicación y cargar un diálogo los resultados saltan a la vista.

FUNCIÓN	USO
AfxGetApp	Obtiene el puntero al objeto CWinApp.
AfxGetInstanceHandle	Obtiene el handle a la instancia actual de la aplicación.
AfxGetResourceHandle	Obtiene el handle los recursos de la aplicación.
AfxGetAppName	Obtiene el nombre de la aplicación.

Tabla 1. Funciones globales de acceso a CWinApp.

```
BOOL CMFCEditorApp::InitInstance()
{
  #ifdef _AFXDLL
          Enable3dControls();
  #else
          Enable3dControlsStatic();
  #endif
  LoadStdProfileSettings();
  CSingleDocTemplate* pDocTemplate;
          pDocTemplate = new
CSingleDocTemplate(
          IDR_MAINFRAME,

RUNTIME_CLASS(CMFCEditorDoc),

RUNTIME_CLASS(CMainFrame),

RUNTIME_CLASS(CMFCEditorView));
AddDocTemplate(pDocTemplate);

  CCommandLineInfo cmdInfo;
  ParseCommandLine(cmdInfo);

  if (!ProcessShellCommand(cmdInfo))
    return FALSE;

  // Código introducido para modificar
color de diálogos

SetDialogBkColor(RGB(255,255,0),RGB
(255,0,0));

  return TRUE;
}
```

En la Figura C se puede ver el orden en el cual se llaman a las funciones *InitInstance*, *Run*, *OnIdle* y *ExitInstance*.

FUNCIÓN MIEMBRO RUN

Esta función ejecuta el bucle "infinito" para tratar los mensajes enviados por Windows. Sólo se sale del bucle, y por tanto se termina la ejecución de la aplicación, al recibir el mensaje *WM_QUIT*. Dentro de este bucle se buscan nuevos mensajes, se preprocesan llamando a la función *PreTranslateMessage* (función sobrecargable), se pasa el mensaje a la ventana correspondiente. Además, si no existen mensajes, se llama a la función *OnIdle* (función sobrecargable) que permite ejecutar acciones en background sin tener que crear otro *thread*. Esta función no suele sobrecargarse, ya que su funcionalidad suele ser suficiente para la mayoría de las aplicaciones Windows. En el ejemplo creado, se ha incluido dos llamadas a la función *AfxMessageBox*, que muestran los puntos donde se empieza y termina el bucle de mensajes.

Los datos internos de la clase CWinApp, permiten almacenar y conocer el estado del objeto de la aplicación en todo momento

```
int CMFCEditorApp::Run()
{
          int Retorno;
          AfxMessageBox("El bucle de
mensajes va a comenzar");
          Retorno = CWinApp::Run();
          AfxMessageBox("El bucle de
mensajes ha finalizado");
          return Retorno;
}
```

FUNCIÓN MIEMBRO PRETRANSLATEMESSAGE

Esta función permite atrapar mensajes dirigidos a la aplicación antes de que ésta los procese. Se pueden implementar filtros de mensajes para realizar determinadas acciones antes de ser procesados por la aplicación. El ejemplo propuesto de sobrecarga de esta función, atrapa el mensaje *WM_LBUTTONDOWN*, produciendo un sonido cada vez que se detecta. Esto se traduce en un pitido cada vez que se pulsa alguna de las partes de la ventana que envía los mensajes a la aplicación, como el caso de la barra de estado o de botones.

```
BOOL CMFCEditorApp::
ProcessMessageFilter (int code, LPMSG
lpMsg)
{
if(lpMsg->message == WM_LBUTTON-
DOWN)
          MessageBeep((UINT)-1);

return CWinApp::ProcessMessageFilter
(code, lpMsg);
}
```

LA FUNCIÓN ONIDLE

Esta función quizá sea una de las más utilizadas o sobrecargadas, ya que permite realizar procesos en background mientras que no se recibe ningún tipo de mensaje. Esta función recibe como parámetro un contador, que indica el número de veces que *OnIdle* ha sido llamada sin que se haya recibido un mensaje, ya que este contador se resetea a 0 cuando se recibe un nuevo mensaje. En esta función se debe devolver TRUE si se necesita volver a procesar o FALSE si no se necesita más proceso en background hasta que se reciba otro mensaje (La figura D ilustra este proceso). Como ejemplo de sobrecarga, se ha creado un timer que produce un beep cada 3 segundos. Para ello, se ha definido una variable en la clase *CMFCEditorApp* que contiene el tiempo anterior, y otra variable local en *OnIdle* que tiene el tiempo actual. Si se detecta una diferencia superior a tres segundos, se produce un pitido y se actualiza el tiempo anterior.

```
// Antes es variable de la clase
CMFCEditorApp

BOOL CMFCEditorApp::OnIdle(LONG
lCount)
{
```

```
        CTime Ahora;
        CTimeSpan Diferencia;

        Ahora = CTime::GetCurrentTime();
        Diferencia = Ahora - Antes;

        if(Diferencia.GetTotalSeconds() >= 3)
        {
                Antes = Ahora;
                MessageBeep((UINT)-1);
        }
        CWinApp::OnIdle(lCount);
        return TRUE;
}
```

Un error un tanto frecuente al sobrecargar funciones de la clase CWinApp es llamar a funciones que emitan mensajes antes de iniciar el bucle de mensajes

Dentro de la función miembro *OnIdle* cabe destacar que esta función no es llamada cuando existe una caja de diálogo modal abierta, ya que no se ejecuta el bucle de mensajes, así como cuando se tiene los menús desplegados. Esto debe tenerse en cuenta para no llevarse ningún tipo de sorpresa cuando se implemente código dentro de esta función. Además, las acciones a tomar en esta función deben ser lo más cortas posibles, ya que acciones prolongadas implican que el bucle de mensajes no se ejecute a la velocidad debida y el sistema no funcione correctamente.

La clase CWINAPP permite inicializar la aplicación y cada una de sus instancias

CONSIDERACIONES ESPECIALES

Un error un tanto frecuente al sobrecargar funciones de la clase *CWinApp* es llamar a funciones que emitan mensajes antes de iniciar el bucle de mensajes, es decir, antes de que se ejecute la función miembro *Run*, lo que se traduce en un funcionamiento incorrecto del sistema.

Además de esto, se recomienda enérgicamente, utilizar las herramientas *ClssWizard* o *ClassBar* para añadir o sobrecargar funciones, ya que estas herramientas crean el esqueleto básico de cada función y sobre todo el modo en el cual se debe llamar a la función de la clase padre.

CONCLUSIÓN

En este articulo se ha pretendido descubrir algunas de las principales, y más utilizadas, funciones y datos de la clase *CWinApp*, lo que va a permitir seguir personalizando las aplicaciones para Windows. Existen más funciones miembro de las que aquí se han expuesto, pero el método de utilización es siempre similar.

En el siguiente articulo se va a tratar otra de las clases fundamentales de la librería MFC como es *CWnd*, clase madre encargada de crear y gestionar todas las ventanas de las aplicaciones, desde los diálogos hasta los propios botones.

NOTAS FINALES

Quiero indicar para los que disponen de conexiòn a la red de redes Internet un par de direcciones donde se puede encontrar informaciòn sobre Visual C ++ 4.0, y todo lo relacionado con Microsoft y sus herramientas. Estas direcciones son:
WEB:
http://www.microsoft.com
http://www.msn.com

Además y si se dispone de conexiòn en la Microsoft Network (MSN), sòlo se tiene que realizar una búsqueda incluyendo Visual C++, o C++ o cualquier otro acrònimo relacionado con el compilador o la programaciòn para Windows y se encontrará multitud de informaciòn técnica y concreta para desarrolladores. Para aquellos que todavìa no dispongan de conexiòn a Internet, podrán también encontrar informaciòn en el Microsoft Roadmap que se entregò en el CD-Rom del número 19 de Sòlo Programadores.

Además comentar que seguimos animando a todos los lectores de Sòlo Programadores y más concretamente a los fieles a esta secciòn a que nos envìen a nuestra redacciòn todo tipo de preguntas, comentarios o dudas sobre este curso monográfico dedicado a la programaciòn en Visual C++ 4.0 . Sòlo Programadores regala todos los meses a sus lectores un paquete Microsoft Visual C++ 4.0 escogiendo una de entre todas las cartas recibidas cada mes, premiándose la agudeza u originalidad de la cuestiòn o comentario. Se ha de indicar el teléfono y direcciòn. Las cartas se han de enviar a:

Editorial Tower Communications
Sòlo Programadores
ref "Concurso Visual C"
C/ Aragoneses, 7
28100 Pol. Ind Alcobendas. Madrid

Para contactar con el autor:

Desde Internet:
E-Mail: jjrio@ibm.net
jjrio@msn.com

Desde MSN: jjrio

CONTENIDO DEL CD-ROM

En el CD-ROM que se adjunta con este ejemplar de Sólo Programadores se ofrece un conjunto de utilidades, editores y compiladores, así como documentación relacionada con el lenguaje de programación Ada.

TUTORIALES

Se incorporan en el CD-ROM dos tutoriales del lenguaje Ada: *Lovelace* y *Ada Tutor*. Aunque se necesita algo más para llegar a dominar el lenguaje en toda su profundidad, ambos tutoriales ofrecen una introducción y una amplia perspectiva, llevando al lector paso a paso con preguntas al final de cada sección además de multitud de ejemplos.

Lovelace es un tutorial en el formato html (formato del World Wide Web de Internet). Para instalar *Lovelace* debe utilizarse la recopilación de la Universidad de Washington (*GWU*) que se encuentra en el directorio \PACKAGES\EZ2LOAD. Para instalar ésta o cualquier otra parte de la distribución habrá que utilizar *GINSTALL.EXE* que se encuentra en ese mismo directorio. La figura 1 muestra la sencilla pantalla de instalación.

Para poder acceder al tutorial *Lovelace* se necesita un navegador de Internet como *Netscape* o *Mosaic*. Si no se dispone de uno se podrá utilizar el navegador *CELLO.EXE* para MsWindows que acompaña a la instalación de *Lovelace* (se encuentra en este mismo directorio). También se entrega en el CD-ROM un navegador para Ms-DOS en modo texto, llamado *LYNX*. En ocasiones es preferible este tipo de navegadores por ser más veloces y distraer menos la atención con elementos no esenciales.

Pueden presentarse problemas para utilizar *CELLO.EXE* debido a que el navegador está diseñado para acceder a páginas a través de la red, con lo que obliga a tener Windows con *WinSock* instalado. Si este no es el caso, se deberá instalar previamente. En el CD-ROM puede encontrarse una versión llamada *Trumpet Winsock* en el directorio \UTL\WINSOCK\ twsk20b.zip. Basta con extraer el archivo *winsock.dll* y copiarlo al directorio en el que se haya instalado el tutorial o al de Windows. Lea atentamente la licencia de uso de *TrumpetWinsock* ya que no es libre (es shareware) y tras el período de prueba habrá que adquirir una licencia para seguir utilizándolo. En el directorio \UTL se encuentran otras utilidades como los descompresores Unzip, Tar, etc.

Para instalar *lynx* simplemente hay que copiar el archivo UTL\DOSLYNX\DLX0_8A.EXE al directorio donde se desee que resida; se auto expandirá al ejecutarlo, generando el navegador y algo de documentación. Si no se dispone de red, para poder utilizarlo únicamente con páginas locales, se debe editar el archivo de configuración *DOSLYNX.CFG* y poner la opción *networked=NO*. Existe también una versión de *Lynx* para Linux, y se puede acceder libremente a su código fuente, aunque esto no ha sido incluido en el CD.

Para ahorrar espacio en el disco duro puede accederse directamente a las páginas del tutorial *Lovelace* que se encuentran extraídas en el directorio \DOCS\LOVELACE. La figura 2 muestra una de las páginas de *Lovelace*.

Otro tutor disponible en el CD-ROM es el *AdaTutor*. Es un tutor para MsDOS en modo texto. Para poder acceder a él es necesario tener un terminal ANSI. En MsDOS bastará con añadir al *CONFIG.SYS* la línea:

DEVICE=C:\DOS\ANSI.SYS

COMPILADORES

El compilador más importante incorporado en el CD-ROM es el *GNAT* (GNU Ada Translator). La licencia bajo la que está protegido es la conocida *GPL* (General Public License) cuyo texto se puede encontrar acompañando al compilador, así como a cualquier otro código *GNU*. Esta licencia permite copiar e instalar el programa en cuantas computadoras se desee y disponer del código fuente así como modificarlo y redistribuirlo; esto siempre que tanto el código como las modificaciones sigan manteniéndose libres, es decir, con los mismos derechos mencionados a cualquiera que se distribuya. Esto no implica

```
The following can be installed from this menu
  1 - GNAT          GNU/NYU Ada 95 compiler (with editor)      (about 20 Meg)
  2 - Editors       AdaCaps and AdaIDE Editors for GNAT        (about 3 Meg)
  3 - Misc          DRGEN and miscellaneous example programs   (about 1 Meg)
  4 - WinHelp       WinHelp Files for Various GCC Tools        (about 2 Meg)
  5 - AdaEd         An Ada 83 compiler with tasking            (about 5 Meg)
  6 - Lovelace      A Hypertext Ada Tutorial                   (about 2 Meg)
  7 - AdaTutor      An Ada tutorial                            (about 2 Meg)
  8 - AdaTUI        A text based UI based on Curses            (about 1 Meg)
  Q - Quit
  Enter Option Number:
```

Figura 1. Pantalla del programa de instalación.

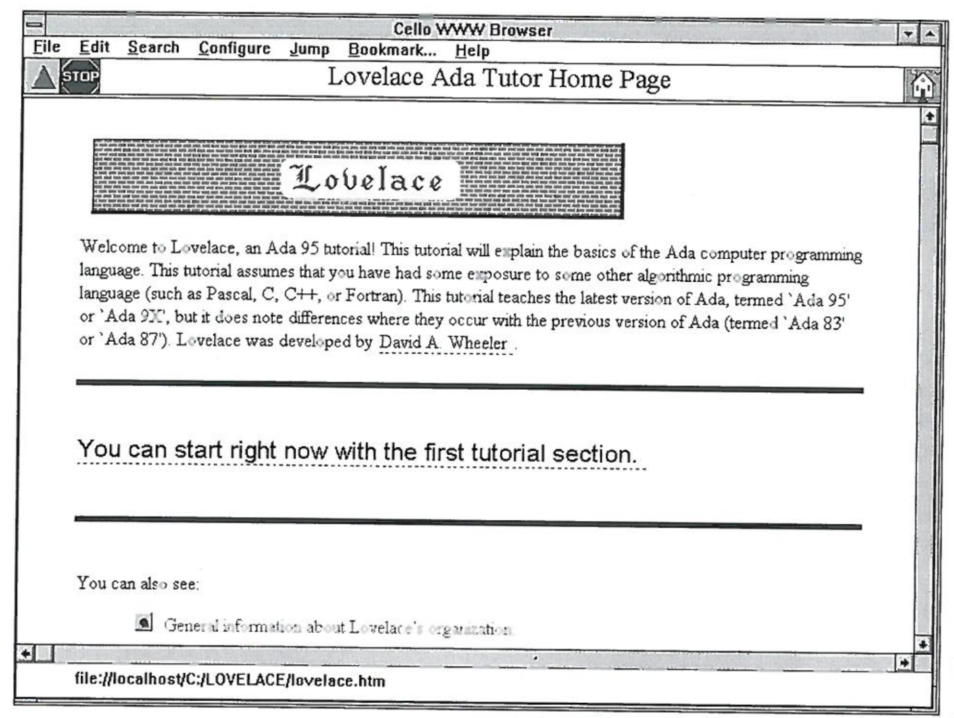


Figura 2. Página del tutorial de Lovelace.

que sea gratis. El desarrollo de *GNAT* ha sido costeado y pagado como cualquier otro producto de software; *GNAT* ha sido financiado principalmente por el Departamento de Defensa de los EE.UU., organismo que financió en un principio el desarrollo de la primera versión del lenguaje, Ada83.

Esta licencia ha permitido disponer de un compilador de Ada de calidad a precio asequible y para multitud de plataformas. Aún así, tiene restricciones; por ejemplo, en MsDOS no se dispone todavía de la parte multitarea debido a las profundas carencias de este sistema. De cualquier manera, es posible aprovechar todas las demás características de Ada para desarrollar software de calidad con *GNAT* para MsDOS.

Para instalar el *port* (versión) de GNAT para MsDOS lo más conveniente es utilizar la recopilación de la Universidad de Washington antes mencionada. Si desea consultar el código fuente o instalarlo para otros sistemas (en la distribución se acompañan versiones para Linux, MsWindows95, NeXT, AIX e instrucciones para obtener la de MsWindowsNT) habrá que mirar en los directorios GNAT2_X y GNAT3_03. En cada caso se acompañan instrucciones específicas.

EDITORES

Con la recopilación de la Universidad de Washington se incluyen dos editores: *AdaCAPS* y *AdaED.* No se trata de simples editores, sino más bien de entornos completos de desarrollo en lenguaje Ada. Desde ellos se puede editar, compilar, ejecutar y depurar el código, así como acceder a la ayuda general o al Manual de Referencia de Ada (*ARM*).

DOCUMENTACIÓN

Se entrega en el CD-ROM una recopilación de las preguntas (con respuestas) más frecuentemente realizadas (Frequently Asked Question) en el formato de las típicas *FAQ* de Internet. Se presenta en formato texto, y partida en varios archivos, en los directorios \DOCS\FAQ.

Empiece por aquí para informarse simplemente sobre qué es Ada, para qué sirve, quién lo utiliza o cualquier otra información general.

Ada es un lenguaje que se basa en una estandarización. La versión anterior del lenguaje es conocida como Ada83 pues en ese año se aprobó como estándar ANSI (en el 87 se estandarizó ISO y por eso en ocasiones, aunque es raro, se le llama Ada87). La revisión que concluye en la segunda versión, la actual, es conocida como Ada95 o simplemente Ada, porque se estandarizó ISO en 1995 (en ocasiones es conocida como Ada9X debido a que se esperaba culminar en esta década aunque no se podía predecir con certeza el año). Los desarrolladores de compiladores se han de ceñir a cada punto del estándar y sólo donde éste proporcione libertad de actuación se encontrarán diferencias entre ellos. Los compiladores han de pasar un proceso de pruebas llamado VALIDACIÓN tras el cual se considera que cumple unos requisitos mínimos y se adapta a la norma. Este proceso no es barato y por ello *GNAT* no está todavía validado en todas las plataformas (el port de *GNAT* para Silicon Graphics sí ha sido validado).

Se incorpora en el CD-ROM copias del estándar ISO de Ada95 directamente en formato accesible de texto (.txt) así como de Postscript (.ps) para imprimir en impresoras láser. También se ofrece una versión en html. Cabe reseñar que el documento *AARM* no es sencillo de leer. No es recomendable como una guía de aprendizaje aunque es el elemento imprescindible de consulta.

El *Rationale* es un documento en el que se razonan cada uno de las decisiones técnicas en el diseño del lenguaje: por qué se aceptan unas medidas, o por qué se incorporan o no unos u otros mecanismos. Podrá encontrar en el CD-ROM el *Rationale* también en formato texto, postscript o html.

En el directorio \DOCS\SUCCESS se dispone de varios documentos que describen éxitos obtenidos en proyectos de gran envergadura utilizando el lenguaje Ada. Estos documentos se encuentran en formato html y en texto plano.

Se puede encontrar las transparencias de una presentación de *GNAT* en el directorio \DOCS\PAPERS en formato Postscript.

Si dispone de acceso a Internet podrá encontrar referencias sobre los lugares más interesantes con información o software sobre el lenguaje Ada en \DOCS\SITES.TXT.



En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relacionado con la programación.

EXTENSIÓN DE FICHEROS EN C++

P *Antes de empezar, felicitarles por el nuevo curso de programación en Visual C++ 4.0 y por el de Visual Basic. Estos dos cursos han conseguido engancharme a la revista. Mis preguntas son si es necesario cambiar la extensión de los ficheros fuentes de .C a .CPP para poder trabajar en C++ y si el curso de Visual C++ será solamente de C o si también incluirá C++.*

José David Calvo Torregosa (Guardamar / Alicante)

R El pequeño programa que nos envía está muy bien y desde luego es un claro ejemplo de cómo utilizar temporizadores desde Visual C.

Vamos a contestar a sus preguntas:
La importancia de la extensión de los ficheros fuente en C es siempre relativa. Si al compilador se le indica el nombre completo, incluida la extensión, le dará exactamente igual el tipo de fuente que contenga. En el caso de que no se incluya la extensión, intentará encontrar un fichero fuente de dicho nombre con la extensión definida por defecto: .C para C y .CPP para C++. El tipo de fuente a compilar se define mediante un parámetro de entrada al compilador o en su defecto, tendrá un lenguaje base, que será el C++ en los últimos compiladores. Además, debe recordar que el concepto de la extensión en los nombres de los ficheros está desapareciendo con los nuevos sistemas operativos que permite cualquier tipo de nombre para un fichero.

Un curso de visual C++ es una empresa muy ambiciosa, que podría ocupar varios tomos. Se ha empezado con partes relativas al C, pero intentará englobar todos los aspectos relativos a este entorno de desarrollo, incluido, como no, el lenguaje C++.

REPRODUCCIÓ DE FICHEROS MIDI

P *Hola, quisiera comentarles un par de asuntos. El primero es que creo que deberían incluir un índice temático sobre los temas ya publicados en la revista, en forma de ficheros HLP para Windows o similar, resultaría muy útil y nos evitaría mucho tiempo de búsqueda para encontrar algún dato. La otra cuestión es referente a cómo podría reproducir un fichero MIDI simultáneamente con la visualización de una demo. Muchas gracias.*

Adolfo Aladro García (Madrid)

R Tiene toda la razón, lo del índice conteniendo la lista de artículos y asuntos tratados, ordenado por temas y por secciones, es una idea que lleva ya mucho tiempo rondando por aquí. En breve se hará realidad y saldrá editado en un CD-ROM con prácticas formas de navegación y búsqueda.

La reproducción de ficheros MIDI como fondo de cualquier animación es una idea fantástica. Su interpretación y reproducción no consume apenas tiempo del procesador ya que es un formato orientado a eventos y no contiene la propia definición de los instrumentos. La explicación sobre cómo reproducirlos excedería las posibilidades de esta sección, pero estamos ultimando un artículo sobre su formato y funcionamiento. La dificultad para su reproducción oscila entre ser extremadamente sencilla al sólo tener que enviar sus datos a un puerto, en las tarjetas que dispongan de entrada MIDI o un poco más complicada en las que es necesario "emular" los instrumentos mediante síntesis FM, como en las tarjetas Sound Blaster clásicas.

SECCIÓN CÓMO HACER UNA DEMO

P *¿Desde qué número se están publicando los artículos referentes a "Cómo hacer una Demo"?. Necesito este dato para posteriormente tratar de conseguirlos. Un saludo.*
Grupo de Programación de Demos y Juegos "Overflow"

(Parla / Madrid)

R La serie de artículos sobre "Cómo programar Demos" comenzó en el número 11 y para obtener los números atrasados, debe ponerse en contacto con nuestro departamento de subscripciones, cuyos datos son:

Departamento de subscripciones de Tower Communications

Srta. Érika de la Riva
Tfno: (91) 661 42 11
Fax: (91) 661 43 86

ASPECTO DE LA PANTALLA EN MODO TEXTO

P *Necesito salvar el aspecto de la pantalla en modo texto, para poder reponerla más tarde, una vez concluida la ejecución del programa que estoy desarrollando. He probado a realizarlo utilizando la BIOS pero resulta muy lento, ¿alguna idea?. Gracias anticipadas.*

Manuel González (Granada)

R Pues sí que existe una buena idea al respecto: la de salvar y reponer la memoria gráfica (sólo la afectada por el modo texto), directamente, sin utilizar los lentos y limitados servicios de la BIOS de vídeo. Esto es muy sencillo y sin ningún "efecto" secundario, tan sólo habrá que seguir los siguientes pasos:

- Definir un *buffer* suficientemente grande para guardar la pantalla. Será el número de caracteres que contenga, multiplicado por dos, ya que cada carácter ocupa dos bytes: uno para el carácter propiamente dicho y otro para sus atributos. Por ejemplo, en el modo tradicional de 80 x 25 ocupará: 80 x 25 x 2 = 4000 bytes.
- Definir un puntero a la memoria gráfica en modo texto. Usualmente es la dirección B8000 en hexadecimal (hay que recordar que en cualquier modo gráfico será la A0000).

- Leer 4000 bytes a partir de esa dirección y guardarlos en el *buffer* anterior.
- Cuando se quiera reponer la pantalla, sólo habrá que volver a copiar el bloque de datos del *buffer* a la memoria de vídeo.

Todo este proceso resulta rapidísimo (tan solo son 4 Kbytes) y no consume apenas tiempo del procesador. Para cualquier otro acceso o cambio en la pantalla también se puede seguir la misma filosofía y aquí se muestran un par de ejemplos:

- Para limpiar la pantalla, escribir 2000 espacios (código ASCII 32), intercalados con 2000 "ceros" para los atributos.
- Para modificar el color de una letra, localizar la dirección de la letra: *B8000 + CoordenadaX + (CoordenadaY * NúmeroCaracteresPorLínea)* y escribir en el byte siguiente el atributo necesario.

PROBLEMAS DE IMPRESIÓN EN RED

P *Hola, un saludo a todos. En mi trabajo tenemos instalada una red local un poco antigua y, aunque tenemos acceso a directorios comunes y podemos compartir ficheros sin problemas, no podemos acceder a impresoras remotas, conectadas en otros ordenadores. Yo necesito imprimir en cierta impresora en color, pero no se me ocurre cómo sin cambiar el software de red instalado. ¿Se puede conseguir mediante algún programa?. A ver que me aconsejan sin tener que rehacer toda la instalación. Un fuerte abrazo a todos.*

Belén Hebra Hinojosa (Madrid)

R Vaya, otro caso de penalidades informáticas ocasionado por una mala instalación o una actualización de software retrasada en exceso. No parece probable que se pueda conseguir el acceso a una impresora remota si la red no lo permite, así que no te molestes en intentar programar los servicios de red. Lo único que se podría hacer en este caso para utilizar la dichosa impresora, es imprimir redireccionando a un fichero ubicado en los discos de red compartidos (en Windows utilizar FILE: en vez de LPT1: o LPT2:, y en DOS utilizando el comando MODE); y desde el ordenador conectado a la impresora copiarlo al puerto adecuado (COPY /B Fichero PRN:). Desde luego no es la panacea, pero evitará el tener que ir con los disquetes de un lado a otro y el tener que trabajar en el otro ordenador.

PROGRAMANDO EN ENSAMBLADOR

P *Tengo un par de dudas respecto a la programación en ensamblador. Primero, quiero programar rutinas rápidas en ensamblador para el modo de vídeo 640 x 480 x 256, pero no sé como funciona este modo. En segundo lugar, veo en muchos programas las instrucciones CLI y STI ; tengo entendido que son para activar y desactivar las interrupciones del sistema, pero no estoy seguro de cuando se deben utilizar. Antes de despedirme, ¿podrían publicar algún artículo adicional sobre Redes Neuronales? Después de leer el primero (número 8) me he quedado con ganas de saber más acerca de ellas.*

Sean Hartnoll (Barcelona)

R Este modo es de lo más sencillo, es similar al habitual 320 x 200 x 256 y su mapeado es "lineal". Esto es, cada pixel ocupa un byte (256 colores = 256 valores posibles) y están almacenados de manera consecutiva: *pixel* 1, *pixel* 2, *pixel* 3 ... Para saber cual es la dirección física de cualquier punto de la pantalla sólo hay que multiplicar la coordenada Y por la anchura de la pantalla (640) y sumarle el valor de la coordenada X. Mediante la dirección obtenida (*offset*) y el segmento habitual (*A000* en hexadecimal), se accede a la *video-ram*. El principal problema reside es que para poder almacenar la pantalla completa (640 x 480 = 307.200 bytes) se necesitan 5 segmentos de 64 Kb (307 / 64), que es el tamaño máximo de la memoria de vídeo accesible por el sistema. Por todo esto, cada vez que se desee escribir (o leer) una línea de pantalla, es necesario comprobar si hay que cambiar de segmento para poder acceder a la dirección correcta y ejecutar el cambio si procede. Para más información sobre cómo se cambia el segmento activo, lo mejor es leer alguno de los artículos que se publicaron en números anteriores de Sólo Programadores.

Tiene razón, las instrucciones CLI y STI son para deshabilitar y volver a habilitar las interrupciones *hardware* del procesador. Se suelen (y deben) utilizar cuando se van a modificar los vectores de interrupción, o las máscaras o flags que permiten o no la generación de ciertas interrupciones, o bien, cuando se está en un proceso crítico de tiempo (lecturas de disco o accesos a puertos) y no se desea que se interrumpa el flujo de la aplicación por alguna "molesta" interrupción. De la misma manera, cuando se modifican los registros importantes del microprocesador, (segmento de código, de pila, etc.), también deben inhabilitarse las interrupciones. Hay que recordar el volver a activarlas, pues si no estaremos bloqueando el sistema con toda seguridad (teclado, ratón, discos ...)

Tomamos en cuenta la petición y se intentará complacerle en breve.